

# THE ANATOMY OF THE 1541



YOU CAN COUNT ON  
Abacus Software



# **THE ANATOMY OF THE 1541 DISK DRIVE**

**A Complete Guide to Using  
The Commodore Disk Drive**

**Authors:** Lothar Englisch  
Norbert Szczepanowski

**Edited by:** Greg Dykema  
Arnie Lee

**ABACUS SOFTWARE**  
P.O. BOX 7211  
GRAND RAPIDS, MI 49510

Second English Printing, June 1984

Printed in U.S.A

Copyright (C)1983 Data Becker GmGH  
Merowingerstr. 30  
4000 Dusseldorf W. Germany  
Copyright (C)1984 Abacus Software  
P.O. Box 7211  
Grand Rapids, MI 49510

This book is copyrighted. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of ABACUS Software, Inc.

ISBN 0-916439-01-1

## PREFACE

The VIC-1541 disk drive represents a very efficient external storage medium for the Commodore user. It is an affordable peripheral. In order to get the most from your 1541, you need the appropriate information. In months of long, detailed work, Lothar Englisch and Norbert Szczepanowski have discovered many secrets of the 1541.

This book progresses from simple storage techniques, to direct access commands, to program chaining techniques. Beginners will welcome the numerous sample programs that are fully explained in clear text. Machine language programmers will particularly like the detailed documentation listing of the Disk Operating System (DOS).

This book contains many useful and ready-to-run programs that need only be typed in. Some of these programs are: routines for extending BASIC, helpful routines such as spooling, efficient address management, a complete household budget planner and an easy-to-use DOS monitor to manipulate individual sectors. Have fun with this book and your VIC-1541 disk drive.

## TABLE OF CONTENTS

<b>Chapter 1: Programming the VIC-1541.....</b>	<b>1</b>
1.1 Getting Started.....	1
1.1.1 The Disk Operating System.....	1
1.1.2 The TEST/DEMO Diskette.....	2
1.1.3 Formatting New Diskettes.....	2
1.1.4 Some Facts about a 1541 Diskette.....	3
1.2 Storing Programs on Diskette.....	4
1.2.1 SAVE - Storing BASIC Programs.....	4
1.2.2 LOAD - Loading BASIC Programs.....	4
1.2.3 VERIFY - Checking Stored programs.....	5
1.2.4 SAVE "@:..." Replacing Programs.....	5
1.2.5 Loading Machine Language Programs.....	6
1.2.6 Storing Machine Language Programs.....	7
1.3 Disk System Commands.....	10
1.3.1 Transmitting Commands to the Disk Drive.....	10
1.3.2 NEW - Formatting Diskettes.....	11
1.3.3 Reading the Error Channel.....	12
1.3.4 LOAD "S",8 Loading the Directory.....	13
1.3.5 SCRATCH - Deleting Files.....	14
1.3.6 RENAME - Renaming Files.....	15
1.3.7 COPY - Copying Files.....	16
1.3.8 INITIALIZE - Initializing the Diskette.....	16
1.3.9 VALIDATE - "Cleaning up" the Diskette.....	17
1.3.10 ? * - The Wildcards.....	18
1.4 Sequential Data Storage.....	20
1.4.1 The Principle.....	20
1.4.2 OPENING a Sequential File.....	21
1.4.3 Transferring Data between Disk and Computer...	24
1.4.4 Adding Data to Sequential Files.....	27
1.4.5 CLOSEing a Sequential File.....	28
1.4.6 Redirecting the Screen Output.....	29
1.4.7 Sequential Files as Tables in the Computer...	30
1.4.8 Searching Tables.....	32
1.4.9 Simple Sorting of Tables.....	35
1.4.10 Mailing List Management with Sequential Data Storage.....	38
1.4.11 Uses for Sequential Storage.....	45
1.5 Relative Data Storage.....	46
1.5.1 The Principle.....	46
1.5.2 The Advantage over Sequential Storage.....	47
1.5.3 OPENING a Relative File.....	47
1.5.4 Preparing the Data for Relative Storage.....	50
1.5.5 Transferring Data.....	52
1.5.6 CLOSEing a Relative File.....	55
1.5.7 Searching Records with the Binary Method.....	55
1.5.8 Searching Records with a Separate Index File...	58
1.5.9 Changing Records.....	61
1.5.10 Expanding a Relative File.....	62

1.5.11 Home Accounting with Relative Data Storage.....	64
1.6 Disk Error Messages and their Causes.....	72
1.7 Overview of Commands with a Comparison of BASIC 2.0 - BASIC 4.0 - DOS 5.1.....	77
<b>Chapter 2: Advanced Programming.....</b>	<b>82</b>
2.1 The Direct Access of any Block of the Diskette.....	82
2.2 The Direct Access Commands.....	86
2.2.1 The Block-Read Command.....	86
2.2.2 The Block-Pointer Command.....	87
2.2.3 The Block-Write Command.....	88
2.2.4 The Block-Allocate Command.....	89
2.2.5 The Block-Free Command.....	90
2.2.6 The Block-Execute Command.....	91
2.3 Uses of Direct Access.....	92
2.4 Accessing the DOS - The Memory Commands.....	94
2.4.1 The Memory-Read Command.....	94
2.4.2 The Memory-Write Command.....	95
2.4.3 The Memory-Execute Command.....	96
2.4.4 The User Commands.....	97
<b>Chapter 3: Technical Information.....</b>	<b>99</b>
3.1 The Construction the VIC-1541.....	99
3.1.1 Block Diagram of the Disk Drive.....	99
3.1.2 DOS Memory Map - ROM, RAM, I/O.....	100
3.2 Operation of the DOS - An Overview.....	104
3.3 The Structure of the Diskette.....	106
3.3.1 The BAM of the VIC 1541.....	106
3.3.2 The Directory.....	107
3.3.3 The Directory Format.....	109
3.4 The Organization of Relative Files.....	114
3.5 DOS 2.6 Rom Listings.....	118
<b>Chapter 4: Programs and Tips For Utilization of the VIC-1541.....</b>	<b>269</b>
4.1 Utility Programs.....	269
4.1.1 Displaying all File Parameters.....	269
4.1.2 Scratch-protect Files - File Protect.....	273
4.1.3 Backup Program - Copying a Diskette.....	278
4.1.4 Copying Individual Files to another Diskette.....	280

4.1.5	Reading the Directory from within a Program...	281
4.2	The Utility Programs on the TEST/DEMO Disk.....	283
4.2.1	DOS 5.1.....	283
4.2.2	COPY/ALL.....	284
4.2.3	DISK ADDR CHANGE.....	284
4.2.4	DIR.....	285
4.2.5	VIEW BAM.....	285
4.2.6	CHECK DISK.....	285
4.2.7	DISPLAY T&S.....	286
4.2.8	PERFORMANCE TEST.....	286
4.3	BASIC-Expansion and Programs for	
	Easy Use of the 1541.....	287
4.3.1	Input Strings of desired Length from the Disk.....	287
4.3.2	Easy Preparation of Data Records.....	290
4.3.3	Spooling - Printing Directly from the Disk....	295
4.4	Overlay Technique and Chaining	
	Machine Language Programs.....	299
4.5	Merge - Appending BASIC Programs.....	302
4.6	Disk-Monitor for Commodore 64 and VIC 20.....	304
<b>Chapter 5:</b>	<b>The Larger CBM Disks.....</b>	<b>317</b>
5.1	IEEE-Bus and Serial Bus.....	317
5.2	Comparison of all CBM Disk Drives.....	319

## Chapter 1: Programming the VIC-1541

### 1.1 Getting Started

There it sits, your new Commodore VIC-1541 disk drive. It's fast and efficient but also intimidating. But have no fear. We will instruct you in the ways of disk programming. The first part of this book gives the beginner an intensive look at the VIC-1541. At least one example follows each command, thereby explaining its functions and capabilities. You will be surprised how easy the operation of your disk drive can be, when you understand the "basics".

The beginner probably uses the disk drive mainly to store programs. Perhaps he has not realized that there are many other ways to use the disk drive. This book attempts to uncover these other ways.

Experienced programmers should not ignore the first chapter. There may be some sections that may shed light on disk usage. This is especially true concerning relative files and data management.

#### 1.1.1 The Disk Operating System

The disk drive is a rather complicated device which coordinates mechanical hardware and electronic circuitry to allow the storage of data on the diskette. When the Commodore 64 or VIC-20 needs to read from or write to the disk drive, it sends commands to the disk drive along the heavy black cable that connects the drive to the computer. The commands sent by the Commodore 64 or VIC-20 are understood at the disk drive by a built in program called the Disk Operating System (DOS).

The DOS is a lengthy program contained on ROM in the disk drive and carries out the activities of the disk drive as commanded by the Commodore 64 or VIC-20. The version of DOS contained in the VIC-1541 carries the designation CBM DOS V2.6.

The Commodore 64 and VIC-20 contain a version of BASIC called COMMODORE BASIC 2.0. Other versions of BASIC (e.g. BASIC 4.0 found of the Commodore 8032) have more advanced disk commands which the VIC-1541 can also understand. In order to use these advanced disk commands, you have to simulate them using BASIC 2.0.

At the end of the chapter is a listing of the BASIC 2.0



## **Anatomy of the 1541 Disk Drive**

commands with corresponding commands of the easier BASIC 4.0, as found on the larger Commodore computers.

### **1.1.2 The TEST/DEMO Diskette**

The VIC-1541 disk drive is packaged with a diskette called TEST/DEMO. Some of the programs contained on it cannot be used without adequate knowledge of the way the disk drive works. For now, lay this diskette aside.

The TEST/DEMO diskette is described in detail later.

### **1.1.3 Formatting New Diskettes**

Brand new diskettes must be prepared before using them to store data. Preparing them is called **formatting**.

What does formatting mean? Each disk drive mechanism has its own special characteristics. A diskette is divided into tracks and information is written along each track (similar to the grooves of a phonographic record). The number of tracks per diskette varies from one manufacturer to another. Each track is divided into sectors, whose number can also vary.

During formatting empty sectors are written to the diskette. A sector is written to each track and sector location and each sector receives its own "address". This allows the DOS to identify its position on the diskette. A sector is also given a code so that the DOS can recognize if this diskette was formatted by this type of disk drive. The code for the VIC-1541 disk drive is 2A. The remainder of the sector (called a block) is used to store data and accommodates exactly 256 characters.

The final purpose of formatting is to construct the **directory** for the diskette. The directory is a "table of contents" of the files stored on the diskette. There is also a special data block (called the bit availability map or BAM) which indicates if a given block on the diskette is already in use or available for use. The directory and BAM are kept on track 18 of the diskette.

#### 1.1.4 Some Facts about a 1541 Diskette

Diskette:

Number of Tracks:	35
Sectors per Track:	17 to 21 (depending on track)
Bytes per block:	256
Total number of blocks:	683
Number of free blocks	644 (the directory occupies the remainder)
Entries in the directory:	144 per diskette

Mechanism:

- intelligent peripheral with its own processor and control system
- connection to serial bus from CBM 64 or VIC-20, device number 4-15 (8 standard)

## 1.2 Storing Programs on Diskette

The most common use of the disk drive is for storage of programs. Storing programs with a disk drive is considerably easier than with a cassette recorder. The greatest advantage of the disk drive is the speed of data transfer to and from the computer. Here's a comparison:

Saving a 3 Kbyte program takes:

- 75 seconds with the VIC-1530 Datasette
- 12 seconds with the VIC-1541 disk drive

An additional advantage is that a diskette can store more programs than the cassette. To load a program, you can consult the directory to view the selection of programs. Even though the cassette drive allows you to store more than one program on a tape, ~~searching~~ for that program is very time consuming.

Before trying any of the following examples in this chapter, you should remember that the diskette must be previously formatted as explained in section 1.3.2 in order to be able to save programs onto it.

### 1.2.1 SAVE - Storing BASIC Programs

Perhaps you previously owned a datasette on which you stored programs. In this case the commands ~~to save~~ programs onto diskette should be familiar to you. The SAVE command for the disk drive is essentially the same as for the cassette drive. You need only tell the computer that the program is to be saved onto the disk drive and not on cassette. This is done by adding the device number (usually 8) to the command SAVE. Normally the drive is preset to respond to this device number. Now write a small BASIC program and save it with the command:

**SAVE"TEST",8**

type in a the NEW command so the program in the computer's memory is erased. In the following section you will learn how the program can be retrieved.

### 1.2.2 LOAD - Loading BASIC Programs

As with the SAVE command, this command is similar to the LOAD command for the datasette with the addition of the device number. Now load in the previously saved program with:

**LOAD "TEST",8**

You can check the program by using the **LIST** command. Any previous program in memory has now been replaced by the program "TEST". It is possible to load a program into the memory without replacing the previous program in memory. Combining two program in memory is called "merging". An example of merging is presented in a later section.

**1.2.3 VERIFY - Checking Stored Programs**

When you have saved a program on disk with the **SAVE** command, it is often desirable to make sure that the program was written error-free. You can do this by using the **VERIFY** command. It has the following format:

**VERIFY"filename",8**

Earlier you saved a program with **SAVE "TEST",8**. This program should still be in memory. Using **VERIFY**, the program in memory is checked against the program stored on diskette. If both programs are identical, the computer responds with **OK**.

To try this out, type a few **BASIC** lines and then give the following commands:

**SAVE "TEST2",8**  
**VERIFY "TEST2",8**

Your computer will respond with **OK** if it is performing correctly.

**1.2.4 SAVE"@:..." - Replacing Programs**

If you try to save your small **TEST** program on the disk again, the computer will respond with a **FILE EXISTS** error and will not complete the **SAVE**. The operating system of the **VIC-1541** disk drive does not allow two programs to be saved under the same name. This is logical because the computer would not be able to distinguish between two programs with the same name.

However you may want to update a program on diskette that was previously saved. There are three ways to accomplish this:

1. Save the program under a different name
2. First erase the old program from the disk and save the new one under the old name

## Anatomy of the 1541 Disk Drive

3. Use the addition @: in front of the file name in the SAVE command

This is used as follows:

**SAVE"@:TEST",8**

If you forget to use the characters @: in front of the filename, and try to save a program whose name is already contained on the diskette, you get the **FILE EXISTS** error.

If you are replacing a program on a diskette then the DOS carries this out as follows:

1. A free block is designated as the first block of the program and its location is stored in the directory entry of the old copy.
2. The new copy of the program is stored in a free area of the diskette.
3. All of the blocks of the old copy are marked as free.

### 1.2.5 Loading Machine Language Programs

Machine language programs are handled a little differently from BASIC programs. A machine language program is transferred to the computer by using a secondary address of 1. When secondary address 1 is used, the program is loaded "absolutely", that is, loaded into memory beginning at the address specified in the first two bytes of the disk file. An example:

**LOAD "MACHPRGM",8,1**

loads the machine language program at an absolute address.

For example, the program may be set up to load at the decimal address 49152, and is started by the command: **SYS 49152**. Should you load a machine language program without the secondary address, you will most likely see the message "SYNTAX ERROR IN ...." if you type **RUN**.

Likewise, trying to **LIST** the machine language program will display nonsense. Unfortunately, machine language programs are not differentiated from BASIC programs in the directory. Both have the file type **PRG**.

Usually, if typing **RUN** results in **SYNTAX ERROR IN .....**, you know that the program is not written in BASIC and should be treated as a machine language program. In this case it must be loaded with the command **LOAD "program",8,1**. It cannot be

started with RUN however! You must first find the execution address of this program.

In a later section is a program that lists all the file parameters of a program. One of the parameters is a load address. This load address is usually the initial execution address of the program and can be called with the command **SYS load address**. You can find the load address of a program with the following program:

```

10 OPEN 1,8,2,"programname,S,R"
20 GET#1,X$:IF X$="" THEN X$=CHR$(0)
30 LB=ASC(X$)
40 GET#1,X$:IF X$="" THEN X$=CHR$(0)
50 HB=ASC(X$)
60 CLOSE 1
70 AD=HB*256+LB
80 PRINT"LOAD ADDRESS: ";AD

```

The program shows the load address of "programname". Here the program file is opened as a sequential data file. The starting address is stored as the first two bytes of the file and read using the GET command and appropriately constructed. The first byte is the low byte and the second byte the high byte of the two-byte address. If the function of this program is unclear, handling sequential files clarified in the next sections.

### 1.2.6 Storing Machine Language Programs

Machine language programs are usually written with an assembler or a machine language monitor and saved using this program. Machine language programs can also be written from BASIC with the individual bytes of the program written in decimal values in DATA statements. A machine language program written in BASIC with the help of DATA statements follows:

```

10 SA=starting address
20 EA=ending address
30 FOR I=SA TO EA
40 READ X
50 POKE I,PEEK(X)
60 NEXT I
80 DATA .....
90 DATA .....

```

In this example, the decimal value of the starting address is placed in line 10 and the ending address in line 20. The decimal values of the individual bytes of the machine language program are typed into the DATA statements of the

## Anatomy of the 1541 Disk Drive

program, separated by commas.

Naturally, you can save any machine language program that you find in this book in the form of a BASIC program. This is, however, a tedious and complicated process. A more elegant and time-saving method is to store the machine language program in true form. This way, you can immediately execute the program after LOADING without requiring any complicated conversion.

The following program will save such a program that is already in memory:

```
10 SA=starting address
20 EA=ending address
30 OPEN 1,8,1,"programname"
40 HB=INT(SA/256):LB=SA-HB*256
50 PRINT#1,CHR$(LB);CHR$(HB);
60 FOR I=SA TO EA
70 PRINT#1,CHR$(PEEK(I));
80 NEXT I
90 CLOSE 1
```

This routine assumes that the machine language program is already in the memory of the computer. If a program is already encoded into DATA statements, the following routine can be used to produce a pure machine language program:

```
10 SA=starting address
20 EA=ending address
30 OPEN 1,8,1,"programname"
40 HB=INT(SA/256):LB=SA-HB*256
50 PRINT#1,CHR$(LB);CHR$(HB);
60 FOR I=SA TO EA
70 READ X
80 PRINT#1,CHR$(X);
90 NEXT I
100 CLOSE 1
110 DATA .....
120 DATA .....
```

Here the addresses and DATA statements are filled in also. The above program writes a machine language program to diskette which can later be loaded with the command **LOAD "programname",8,1**. Then the program can be executed with the command: **SYS (starting address)**. Machine language programs can also be loaded and executed from a BASIC program. Such a program might have this form:

```
10 IF A=0 THEN A=1:LOAD"programname",8,1
20 SYS (starting address)
```

The IF command in line 10 is puzzling at first. It must be present because after performing a LOAD from within a program, the BASIC interpreter begins executing again at the

first line of the new BASIC program. Because the machine language program doesn't usually overlay the BASIC program in memory, the original BASIC program remains intact and is therefore is re-executed. If you use the routine:

```
10 LOAD"programname",8,1
20 SYS (starting address)
```

the program continues to LOAD "programname" again, and the SYS command is never executed. If the variable A is present, the program branches to line 20 at the end of the first command on line 10. This loader can be placed on the diskette together with the machine language program. To execute the machine language program, you need only give the commands:

```
LOAD"loader",8
RUN
```

This has the advantage that the starting address of the machine language program need not be known, because it is included in the SYS of the loader.



## Anatomy of the 1541 Disk Drive

### 1.3 Disk System Commands

As already mentioned, the VIC-1541 disk drive is similar to the the earlier, larger disk drives of the Commodore family - the CBM 4040, 8050, 8250. They are all intelligent peripheral device with their own processor and control system. The Disk Operating System (DOS) occupies no space in the memory of the Commodore 64 or VIC-20 and yet offers a flexible set of efficient commands. These commands effectively expand the builtin commands of your Commodore computer.

Because the disk drive is an intelligent peripheral, the commands of the DOS can be executed independently of the computer. But because the commands are not found in the version of BASIC supplied in the Commodore 64 or VIC-20, you will have to communicate to the disk using a special method. When the commands are sent to the disk drive, the DOS interprets and carries out the desired task.

#### 1.3.1 Transmitting commands to the Disk Drive

Commands intended for the disk drive, are sent over a **channel**. You can communicate with the disk drive over any of the 15 available channels. But channel 15 is reserved as the **command channel**. Data transfer over this channel takes place as follows:

- opening the channel (OPEN)
- data transfer (PRINT)
- close the channel (CLOSE)

In the OPEN command you specify a logical file number (arbitrary between 1 and 127), a device number of the disk drive (usually 8) and the secondary address (15 for the command channel). You can also send a command to the device as illustrated below:

```
OPEN lfn,8,15,"command"  
or  
OPEN lfn,8,15:PRINT#lfn,"command"
```

The number 8 is the device number of the disk drive and the number 15 is the secondary address or channel number. The parameter lfn is the logical file number which is used in subsequent commands (PRINT#, INPUT#, GET#). It can be a number in the range 1-127. The **"command"** can either follow the OPEN statement directly, or can be transferred with a PRINT# command following the opening. Any number of system commands can be transmitted until the channel is closed, but must be referenced by the logical file number used in the OPEN command.

### 1.3.2 NEW - Formatting Diskettes

The command to format a diskette is called **NEW** and can, as every other command, be abbreviated to its first letter (**N**). As already mentioned, the command can follow an **OPEN** command or be given in a **PRINT#** command. The **NEW** command has the following format:

**NEW:diskname,id**

The parameter **diskname** may contain up to 16 characters and is stored in the header of the diskette directory. The parameter **ID** (identification) consists of two arbitrary characters, so that the DOS can recognize if a different diskette has been used. Since you can freely choose the **id**, this allows you to uniquely identify each diskette. Here is an example for formatting a disk:

**OPEN 1,8,15,"NEW:ABCDISK,KL"**

The command can be abbreviated to:

**OPEN 1,8,15,"N:ABCDISK,KL"**

You need only use the command once - when you first use a brand new diskette. Formatting takes about 80 seconds. Formatting uses the processor of the 1541 drive while the processor of the computer is not needed; you can continue to work with the computer.

To use the command with a **PRINT#** statement, the following commands must be given:

**OPEN 1,8,15**                      to open the channel  
**PRINT#1,"N:ABCDISK,KL"**

The number 1 in the **PRINT#** command is the logical file number corresponding to the **OPEN** command. Other commands may also be transmitted over this channel after the **PRINT#** statement. When no more commands are to be transmitted, the channel must be closed. This is accomplished through the use of the **CLOSE** statement. Give the following command after formatting:

**CLOSE 1**

Now the command channel is closed. The number 1 is again the logical file number of the corresponding **OPEN** command.

## Anatomy of the 1541 Disk Drive

### 1.3.3 Reading the Error Channel

When the Commodore 64 or VIC-20 is incorrectly programmed, it responds with an error message. Disk commands are carried out and verified by the processor of the disk drive. Therefore the computer cannot directly display error messages that are detected by the disk drive. Errors are indicated by the flashing red LED on the disk drive. In order to determine which error has occurred, the computer must read the error from channel 15. Therefore channel 15 must be OPENed, if this has not already been done. Then the error can be read with the INPUT# command. An error is sent back to the computer in four fields -

Field 1: Error number  
Field 2: Description of the error (string)  
Field 3: Track number  
Field 4: Sector number

The track and sector information may indicate where the error occurred (if these fields are relevant to the command). These four fields of the error message must be read into four variables. You can use an INPUT# statement followed by four variables. An example of reading the error channel:

```
OPEN 1,8,15          (if not already done)
INPUT#1,EN,DES,TR,SE
CLOSE 1
```

The INPUT# statement must be entered from within a program. It is not proper to issue an INPUT# statement from command mode.

```
10 OPEN 1,8,15
20 INPUT#1,EN,DES,TR,SE
30 PRINT EN;DES;TR;SE    (to display the error)
40 CLOSE 1
```

To understand the operation of this program, first create the following error:

```
OPEN 1,8,15,"NEW ABCDISK,T1"
CLOSE 1
```

When you have given these commands, the red LED on the disk drive begins to blink. Did you spot the error? A colon is missing from the command NEW. Now type the program to read the error channel and type RUN. The error will appear on the screen:

```
34 SYNTAX ERROR 0 0
```

The 34 is the number of the error, which is explained later. The track and sector fields are 0 because this information

is not relevant to this error.

If you read the error channel when an error had not occurred, the message:

```
0 OK 0 0
```

is returned. In any case, if the red LED on the drive blinks, check the syntax of the command, since most errors can be easily recognized. Otherwise, you can simply read the error channel to find the error which the DOS has detected. A detailed description of the error message and their causes follows in section 1.6.

### 1.3.4 LOAD "\$",8 - Loading the Directory

The **directory** is a "table of contents" of the diskette. All the files on the diskette are cataloged here. Be sure to note that loading the directory has a disadvantage: any program previously in memory is overlaid by the directory information. The directory is loaded by typing:

```
LOAD "$",8
```

and can be viewed with the **LIST** command. Try **LOADing** the directory of the TEST/DEMO diskette that accompanies your disk drive. Insert this diskette into the disk drive and enter: **LOAD "\$",8** to load the directory. Then display the directory by using the **LIST** command. What follows should be shown on the screen

```
0      "1541test/demo      " zx 2a
13     "how to use"        prg
5      "how part two"      prg
4      "vic-20 wedge"      prg
1      "c-64 wedge"       prg
4      "dos 5.1"          prg
11     "copy/all"         prg
4      "disk addr change" prg
4      "dir"              prg
6      "view bam"         prg
4      "check disk"       prg
14     "display t&s"      prg
9      "performance test" prg
5      "sequential file"  prg
13     "random file"      prg
```

A lot of information is kept in the directory. Let's look at the first line, the header of the directory. The number **0** in this line means that the directory is of the diskette in drive 0. Other disk drives such as the 4040, contain two disk drives - drive 0 or drive 1. On the 1541 the drive

## Anatomy of the 1541 Disk Drive

number is always 0. Next follows the name and ID of the diskette as set up by formatting. The characters 2A symbolize the disk format. If this format is not 2A then this diskette was not formatted with a 1541 drive.

Next are the individual file names, their lengths in blocks in the first column and the file type in the last column. This diskette contains three different file types:

PRG These are PROGRAM files, written in either BASIC or machine language

SEQ Sequential data files, explained later

REL This is another form of data storage, also explained later

The length of the files is given in blocks. Each block contains 256 bytes. You can find the approximate size a program, by subtracting 2 bytes from each 256-byte block that the file occupies. Finally at the end of the directory is the number of free blocks remaining on the disk. When you add the lengths of the files and the number of free blocks, the result is the total number of available blocks on a diskette (664).

If you own a printer, this directory can be printed as you would print a program listing. Use the following commands:

OPEN 1,4	open the printer
CMD 1	the printer is now linked to the screen
LIST	the directory will be printed
PRINT#1	send a RETURN to the printer
CLOSE 1	close the printer again

It is assumed that the directory is already loaded with the **LOAD"\$",8** command before these commands are executed. By inserting a wildcard when loading the directory, you can cause only part of the directory to be loaded, such as only the programs. This is explained in section 1.3.10

### 1.3.5 SCRATCH - Deleting Files

Sometimes an unneeded file must be removed from the diskette. The SCRATCH command is provided for doing so. Before using this command, you must be sure that the name given in the SCRATCH command corresponds with the file to be deleted. An unintentionally deleted file can ruin many hours or even days of work, so be careful before using the SCRATCH command.

To delete a file, the following format should be used:

```
PRINT#lfn,"SCRATCH: filename1, filename2,..."
```

More than one file can be deleted by using a single command. But remember that only 40 characters at a time can be sent over the transmission channel to the disk drive.

For example, to erase a file with the name TEST, the following commands are used:

```
OPEN 1,8,15,"S:TEST"
CLOSE 1
```

If channel 15 is already open, only the PRINT# command is required:

```
PRINT#1,"S:TEST"
```

It is possible to delete the entire contents of a diskette. This is discussed in section 1.3.10, the **wildcard** character (\*):

```
PRINT#1,"S:*"
```

But be very careful! Make sure that you do not need any of the files on the diskette before using this command. After completing the operation the error channel transfers the message:

```
01 FILES SCRATCHED nn 00
```

where nn is the number of deleted files. This message can be read with the routine given in section 1.3.3.

### 1.3.6 RENAME - Renaming Files

You can also change the name of a file on the diskette. The command RENAME is provided for this purpose. It has the following format:

```
RENAME:newname=oldname
```

For example, if you want to change the name of the file from TEST to PEST you would use the following commands:

```
OPEN 1,8,15,"R:PEST=TEST"
CLOSE 1
```

or

```
OPEN 1,8,15
PRINT#1,"R:PEST=TEST"
CLOSE 1
```

Note that you cannot rename a file until it is CLOSED.

### 1.3.7 COPY - Copying Files

Using this command, a file can be copied on a diskette. Several different sequential files can be used to create a new file. If, for example, you have a data record for each month of your household expenses and they have the names EXP.01, EXP.02, etc. you can combine them into quarters (EXP.Q1 for example) with this command. The COPY command has the format:

```
COPY:newfile=oldfile1,oldfile2,...
```

So, the named data records can be combined as follows:

```
OPEN 1,8,15,"C:EXP.Q1=EXP.01,EXP.02,EXP.03"
CLOSE 1
```

This method of combining data records **cannot** be used for programs. Only a single program can be copied on the diskette. Also the name of the new file must not already exist on the diskette.

The COPY command is seldom used. This is because copying files onto the same diskette usually makes no sense. The only sensible use of the command is to combine several sequential or user files into a single file.

Copying files from one diskette to another diskette is much more sensible. This is indispensable for data security. If you own two disk drives, you can assign the device number 9 to one of them and use the program COPY/ALL to copy files from one to the other. This program is found on the TEST/DEMO diskette.

We have also thought of you who have only one disk drive. A utility program is included in section 4.1 to allow you to copy individual files and even the entire diskette.

### 1.3.8 INITIALIZE - Initializing the Diskette

The DOS requires a BAM (Block Allocation Map) to be present on each disk. The BAM is a layout of the usage of the blocks on each diskette. It marks each block on the diskette

as free for use or allocated (already in use). If you change diskettes in the drive and the new diskette has the same id as the old diskette, the DOS will not recognize the fact that you have changed diskettes. The BAM of the new diskette will be different, but the DOS will still be working with the old BAM.

Therefore, each diskette should be given a unique id when you format it. It is a good practice to give each diskette a different id. You can force the disk drive to read the BAM of a new diskette by issuing the **INITIALIZE** command. This command has the following format:

```
PRINT#1fn,"INITIALIZE"
```

or shortened to

```
PRINT#1fn,"I"
```

Example:

```
OPEN 1,8,15,"I"  
CLOSE 1
```

If you change diskettes and also change data records, then we strongly recommend that you use the **INITIALIZE** command after changing the diskettes, to be safe.

### 1.3.9 **VALIDATE** - "Cleaning Up" the Diskette

The command **VALIDATE** frees all allocated blocks that are not assigned to normally **CLOSED** files. For example, if you **OPEN** a file, and transfer data to that file, but forget to **CLOSE** the file, the **VALIDATE** command can be used to free the data blocks that were written to. If you use the direct access commands, be sure to allocate them (using the **BLOCK-ALLOCATE** command) or the **VALIDATE** command will free them again.

The command has an additional function: If a file is deleted using the **SCRATCH** command, the file type in the first byte of the file entry is set to 0. It no longer appears in the directory. If you now change this byte back to its old file type with the DOS monitor (described later) or other direct access commands, **VALIDATE** will restore the file. If it has not been overwritten, it will be the same as before the **SCRATCH** command. The command has the following format:

```
PRINT#1fn,"VALIDATE"
```

or the shorter form

```
PRINT#1fn,"v"
```



## Anatomy of the 1541 Disk Drive

An example:

```
OPEN 1,8,15,"V"  
CLOSE 1
```

If you have a diskette such that the sum of the file lengths plus the number of free blocks does not equal the total number available (664), use the **VALIDATE** command to restore it.

Another example: If you want to store a program or data record that uses more than the number of free blocks, the DOS will give the error **DISK FULL**. If the disk had shown some blocks free before, the number is now zero. The **VALIDATE** command will restore the original free blocks.

### 1.3.10 ? \* - The Wildcards

There are two wildcard characters - the asterisk (\*) and the characters of the first file on the disk that begins with the characters which precede the asterisk. An example:

```
LOAD"TEST*",8
```

This command loads the first program that begins with the first four letters "TEST". The command:

```
LOAD"**,8
```

loads the first program on the diskette because there are no characters in front of the asterisk. The asterisk in the **SCRATCH** command has a different effect. If used in the **SCRATCH** command, not only the first file will be deleted, but all files. For instance, the command:

```
OPEN 1,8,15,"S:TEST*"  
CLOSE 1
```

erases all files beginning with the the letters "TEST". This must be taken into account! Loading the directory with an asterisk can also select certain files. An example:

```
LOAD"$A*",8
```

loads only the directory of the files that begin with the letter "A".

The DOS offers an additional use of the asterisk that has not been mentioned yet. It can also select file types if the asterisk is followed by the first letter of the desired file type. Here is a summary:

```
*=S      selects only sequential files
*=P      selects program files
*=R      selects relative files
*=U      selects user-files
```

For example, the command:

```
LOAD "$*=P",8
```

causes only the directory entries of programs to be loaded and shown when you type **LIST**. This can also be used with the **SCRATCH** command to delete all sequential files, for instance. Here is the command:

```
OPEN 1,8,15,"S:*=S"
CLOSE 1
```

With the question mark, certain characters of a file name can be declared "not relevant". To illustrate the function of the question mark, here are two examples of shortened file names and their effects:

```
A?????    - refers to a six-letter filename of which
             first character is A
????TEST   - refers to an eight-character filename, the
             last four letters of which are TEST
```

A combination of asterisks and question marks is allowed. You should notice, however, that an asterisk followed by question marks has no meaning. Two examples of combinations of asterisks and question marks:

```
????.*    - refers to all file names that have four
             characters before a period
TEST.??*   - refers to all file names having at least 7
             characters, of which the first five are
             TEST.
TEST-??01*=S - refers to all sequential files whose names
             have at least nine characters, the first
             five being TEST- and the eighth and ninth
             being 01
```

## Anatomy of the 1541 Disk Drive

### 1.4 Sequential Data Storage

A disk drive need not be used exclusively for storing programs. If you have written a program that manages a large quantity of data, you need a fast way of organizing it. Sequential data storage is not the fastest, but it is the easiest method of managing data. This method is comparable to sequential storage on a cassette, which can be maintained in a program as such:

1. Load the program
2. Read the entire data file into the memory of the computer
3. Work with the data in memory (change, delete, combine)
4. Write the new file on an external medium (cassette, diskette)
5. Exit the program

The maximum number of data items that the program can handle depends on the size of the computer's memory, because a single data item cannot be changed or erased directly on the cassette or diskette. To that end, the entire set of data items must be read in, changed, and then rewritten again. Reading and rewriting the data occurs remarkably faster on a disk drive than on cassette.

It is worth mentioning that programs which work with sequential data on cassettes can be easily modified to work with disk. Only the corresponding OPEN commands need be changed.

#### 1.4.1 The Principle

A sequential data file consists of several data records that are further divided into fields. The following is a name and address file and illustrates the principle of sequential data storage. Individual names and addresses comprise the data records of this file. A record consists of several fields (last name, first name, etc.). The structure of the file looks something like this:

```
=====
Field 1 : Field 2 : Field 3 : Field 1 : Field 2 : Field 3 :
=====
      Data record 1           :           Data record 2
-----
                          FILF
-----
```

Only two records are shown above. The data records of a file are stored one after another (sequentially) as are the fields within each record. The fields and records may be of any length. For example, field 1 of record 1 may be longer than field 1 of record 2. This is possible because the fields are separated from each other by a special character (the RETURN character), which is generated by the PRINT# statement. When read back into the computer by the INPUT# statement, the RETURN character is recognized as a field separator.

Each field is associated with a variable when written with a PRINT# statement or read with an INPUT# statement.

How does the computer know, when reading the data, where each field ends? Each field ends with a RETURN character. The RETURN character has the decimal ASCII value 13. An example of a telephone directory file illustrates this. Our telephone directory file has three fields:

```
FIELD 1 : LAST NAME
FIELD 2 : FIRST NAME
FIELD 3 : TELEPHONE EXTENSION
```

Let's look at a section of this previously written file (the character + symbolizes a RETURN):

```
Position:      1111111111222222222233333333334444444
                1234567890123456789012345678901234567890123456
                -----
Data:          SMITH+JOHN+236+LONG+TIM+121+HARRIS+SAM+654+...
                -----
```

You can see that the fields are of different lengths and are all separated by a RETURN character. This RETURN character is automatically written after the data field by a PRINT# statement, provided the PRINT# statement is not followed by a semicolon (which suppresses the RETURN character).

These data items are assigned to the variables with an INPUT# statement. After that, another INPUT# must follow in order to read the next field, and so on. The following sections explain the fundamentals of writing programs using sequential data storage.

## 1.4.2 Opening a Sequential Data File

To create a sequential data file, you must first OPEN the file. When opening a file to be written to, the following is carried out:

1. The diskette is checked to see if an existing file has

## Anatomy of the 1541 Disk Drive

the same name. If so, the error message **FILE EXISTS** is given by the DOS.

2. The file entry in the directory is written. In the file type it is noted that this file is not yet **CLOSED**. This appears in a directory listing with an asterisk which precedes the file type.
3. A free block is found, into which the first data items are written. The address (track and sector) of this free block is stored in the file entry of the directory.
4. The number of blocks in the file is set to 0, because no blocks of the file have been written yet.

The **OPEN** command specifies for what purpose (mode) the file is to be used (reading or writing). The format of the **OPEN** command looks like this:

**OPEN lfn,8,sa,"filename,filetype,mode"**

When the logical file number is between 1 and 127, a **PRINT#** statement sends a **RETURN** character to the file after each variable. If the logical file number is greater than 127 (128-255), the **PRINT#** statement sends an additional line-feed after each **RETURN**. This is necessary for printers, for example, that do not provide an automatic line-feed after a **RETURN** character.

The secondary address (sa) can be a value between 2 and 14. The secondary address indicates the channel over which the computer is to transfer data to and from the disk drive. Secondary addresses 0 and 1 are reserved by the DOS for saving and loading programs. Secondary address 15 is designated as the command and error channel. Should several files be open at once, they must all use different secondary addresses, as only one file can use a channel. If, however, a file is opened with the secondary address of a previously opened file, the previous file is closed.

A maximum of 3 channels can be opened with the VIC-1541 at a time. When utilizing relative data files, the DOS requires 2 channels per file. Therefore, the following maximum combinations are possible:

- 1 relative and 1 sequential file
- or - 3 sequential files

When specifying the filename to be written to (in the **OPEN** command), you must be sure that the file name does not already exist on the diskette. If a file that already exists is to be opened for writing, an at sign followed by a colon (@:) must be placed in front of the file name (same as in the **SAVE** command). For example:

**OPEN 1,8,2,"@:ADDRESSES,S,W"**

The file type must be given when the file is opened. The file type may be shortened to one of following:

- S - sequential file
- U - user file
- P - program
- R - relative file

User files are sequential files that are listed in the directory with the file type **USR**. It is not a data file in the true sense. This file type is usually used when output that normally goes to the screen (BASIC listing, directory) is sent to the disk. In section 1.4.6 you find a description of this technique.

The last parameter (mode) establishes how the channel will be used. There are four possibilities:

- W - Write a file (WRITE - section 1.4.3)
- R - Read a file (READ - section 1.4.4)
- A - Add to a sequential file  
(APPEND - section 1.4.4)
- M - read a file that has not been closed  
("discovered" by us in the DOS listing and explained in section 1.4.5)

Now open a sequential file with the name **SEQU.TEST** for writing:

**OPEN 1,8,2,"SEQU.TEST,S,W"**

If you now load the directory with **LOAD"\$",8** and then **LIST** it, you see this file listed with an asterisk before the file type:

```
0      SEQU.TEST      *SEQ
```

But you are no longer allowed to close this file! After a file is **OPENed** and data written to it, it must be closed before the directory is loaded!

While a file is open, the command/error channel 15 may be opened, but when channel 15 is closed, all other channels are closed as well. You must take note of this.

Now some examples of the **OPEN** command:

```
OPEN 1,8,2,"SEQU.TEST,S,R"    - open a sequential file for
                                reading
OPEN 2,8,3,"SEQU.TEST,U,W"    - open a user file for writing
OPEN 3,8,4,"TEST,P,R"         - open a program file for
                                reading
```

## Anatomy of the 1541 Disk Drive

```
OPEN 4,8,5,"SEQU.TEST,S,A"    - open a sequential file for
                                appending data
OPEN 5,8,6,"CSTMRS.1983,S,M"    - open the unclosed customer
                                file for reading
```

### 1.4.3 Transferring Data Between Disk and Computer

After opening a file for writing, you transfer data to be stored to the diskette with the PRINT# statement. This statement transmits an additional RETURN that is required for separating data. In the following example, a file is OPENed, data written to it, and CLOSED again. PRINT# can also be used as a direct command, that is, outside of the program, so the following commands can be typed one after the other and executed. Now open a file with the name "TEST":

```
OPEN 1,8,2,"TEST,S,W"
```

You should notice that the red LED on the disk drive was lit. It signals the fact that a file was OPENed. You can now write to the file named TEST. Here is how we would write a name and address record consisting of 4 fields:

```
PRINT#1,"SAM"
PRINT#1,"HARRIS"
PRINT#1,"2001 MAIN STREET"
PRINT#1,"ANYTOWN"
```

Now these data items have been written to the file so we can close the file with CLOSE 1. The red LED should go out. In order to read this data again, you must open the file in the read mode (R). Because the INPUT# statement cannot be used directly, a small program must be written:

```
10 OPEN 1,8,2,"TEST,S,R"
20 INPUT#1,FN$
30 INPUT#1,LN$
40 INPUT#1,ST$
50 INPUT#1,CT$
60 CLOSE 1
70 PRINT"FIRST NAME: ";FN$
80 PRINT"LAST NAME: ";LN$
90 PRINT"STREET: ";ST$
100 PRINT"CITY: ";CT$
```

The program is simple to explain:

```
Line 10      The file TEST is opened for reading
```

Lines 20-50      The data are read in the same order as they were written. Variables are used so that the data can be printed later.

Line 60            The file is closed.

Lines 70-100      The data are printed out on the screen.

When you enter this program and type RUN, the data will appear as written earlier, on the screen:

```
FIRST NAME:  SAM
LAST NAME:   HARRIS
STREET:      2001 MAIN STREET
CITY:        ANYTOWN
```

Four INPUT# statements were used to read the data because the name and address record is composed of four fields. But when a record is written that has, say, 20 fields, it is very time-consuming to type out 20 INPUT# statements. A loop can make this much simpler. This is obvious in this example:

```
10 OPEN 1,8,2,"TEST,S,R"
20 FOR I=1 TO 4
30 INPUT#1,D$(I)
40 NEXT I
50 CLOSE 1
60 PRINT"FIRST NAME: ";D$(1)
70 PRINT"LAST NAME: ";D$(2)
80 PRINT"STREET: ";D$(3)
90 PRINT"CITY: ";D$(4)
```

Here, instead of four separate string variables, an array with index 1-4 is used. It should be noted that in BASIC 2.0, if an index higher than 10 is used, the array must be dimensioned with a DIM statement. Should we want to read in 20 fields, the statement DIM D\$(20) must be given before any are read.

There are still more ways of shortening input and output of data. With the INPUT statement for keyboard input, several variables can be given in one line, separated by commas. For example:

```
INPUT FN$,LN$,TE
```

With this statement, three variables must be entered, such as:

```
NICHOLAS,MULLER,7465
```

The read data can be printed on the screen with:

```
PRINT FN$,LN$,TE
```



## Anatomy of the 1541 Disk Drive

In this manner, sequential data can be written and later read back in again. The only difference is that the string variables containing the data to be written must be separated by commas enclosed in quotes. For example, if you wish to write the previous variables to a file, the PRINT# statement command must be changed as follows:

```
PRINT#1,FN$,"LN$","TE
```

Numeric variables need only be separated with a comma from the other variables. To read the data, use the command:

```
INPUT#1,FV$,LN$,TE
```

Because the maximum number of characters read by an INPUT# statement may not exceed 88, this method of reading is only marginally useful. If a field in a record is more than 88 characters long, a different statement must be used. This is the GET# statement, which reads each individual character, one at a time. Suppose you want to read a record of which a field is 100 characters long. This record can be placed in a string variable with the following routine:

```
10 OPEN 1,8,.....
20 D$=""
30 FOR I=1 TO 100
40 GET#1,X$
50 D$=D$+X$
60 NEXT I
70 GET#1,X$
80 CLOSE 1
```

At the end of this program, the string variable D\$ will contain the 100 characters of the data field. After opening a sequential data file, the DOS establishes a pointer that always points to next character to be read. We assume that the data was written with a PRINT# statement without a trailing semicolon, so that a RETURN was written at the end of the data item. After reading the first 100 characters, the pointer points to this RETURN. The next GET# in line 70 is necessary to read the RETURN found at the end of the field. Then the next GET# statement can read the next field and not the RETURN.

In the above example, we used data records with a constant length of 100 characters. According to the rules of sequential access, the length of data records need not be constant. Since the INPUT# statement can only read a maximum of 88 characters, we will use the GET# statement to recognize the RETURN as the end of a field. Such a routine looks like this:

```
10 OPEN 1,8,.....
20 S$=""
30 GET#1,X$
40 IF X$=CHR$(13) THEN 80
```

```

50 S$=S$+X$
60 IF ST<>64 THEN 30
70 CLOSE 1:END
80 PRINT S$
90 GOTO 20

```

Here a file with variable record length is read and printed on the screen. Naturally, you can use the data in other ways instead of printing it on the screen.

To avoid the problem of reading data records of more than 88 characters, divide the record into several parts, which you can combine after reading them.

#### 1.4.4 Adding Data to Sequential Files

If you want to add data to a sequential file, you have to read the entire file into memory, add the data, and write the new file back to the diskette again. This is a very time-consuming process. For this reason, the DOS offers an easier alternative to add to a sequential data file without reading the entire file. This is made possible through the OPEN mode **A** (Append). If you have a sequential data file, as in the previous section, you can add data to it by selecting the **A** mode in the OPEN command. An example follows.

Give the following commands:

```

OPEN 1,8,2,"TEST2,S,W"
PRINT#1,"1. DATA RECORD"
CLOSE 1

```

Now you have a sequential data file containing one data record. This file can be expanded with two more records as follows:

```

OPEN 1,8,2,"TEST2,S,A"
PRINT#1,"2. DATA RECORD"
PRINT#1,"3. DATA RECORD"
CLOSE 1

```

Now the file **TEST2** has three data records. You can check this with the following program:

```

100 OPEN 1,8,2,"TEST2,S,R"
110 FOR I=1 TO 3
120 INPUT#1,DR$
130 PRINT DR$
140 NEXT I
150 CLOSE 1

```

After the program starts, the data records is read and printed on the screen.

## Anatomy of the 1541 Disk Drive

You can see that the append A mode makes it quick and easy to expand a sequential data files.

### 1.4.5 Closing a Sequential File

OPENed data files can be closed with the CLOSE command. This command has the format:

**CLOSE lfn**

The parameter **lfn** is the logical file number of the file that was used in the OPEN statement. Should several files need to be closed a CLOSE statement must be given for each one. When the last file is closed, the red LED on the drive goes out.

As you already know, data is sent to the disk drive over a channel. This channel uses storage inside the disk (called a buffer) in which the data transmitted by the computer is stored. When this buffer is full, its contents are written to the diskette.

When the file is closed, any data still in the buffer is written to the diskette. An unclosed file is incomplete and is also not recognized by the DOS as a properly closed file. The DOS allows no read access in the R (Read) mode and responds **WRITE FILE OPEN** when trying to read an unclosed file.

This could be a problem if the DOS did not allow read access to a file. For this reason, the DOS offers the M mode. A file that is marked as an improperly closed file can be read in this mode. It is logical to then write these records to a second file which can then be properly closed. In this way one can "rescue" a file.

The following program will transfer an improperly closed file (original file) to a correctly closed file (destination file):

```
100 INPUT"ORIGINAL FILE NAME";SS
110 INPUT"DESTINATION FILE NAME";DS
120 OPEN 1,8,2,SS+","S,M"
130 OPEN 2,8,3,DS+","S,W"
140 INPUT#1,X$
150 PRINT#2,X$
160 IF ST<>64 THEN 140
170 CLOSE 1:CLOSE 2
180 OPEN 1,8,15,"S:"+SS
190 CLOSE 1
```

At the completion of the program, the unneeded original file

is deleted (scratched).

#### 1.4.6 Redirecting the Screen Output

Any output appearing on the video screen (PRINT, LIST, etc) can be redirected to a sequential data file. This is accomplished through the CMD command, which has the following format:

##### **CMD lfn**

For this to occur, a file of type **USR** must be opened. To transfer a BASIC program listing, for instance, as a sequential file on diskette, use the following commands:

```
OPEN 1,8,2,"TEST.LIST,U,W"
CMD 1
LIST
CLOSE 1
```

The command **CLOSE 1** causes further output to be sent to the screen.

Storing a program as a sequential file on disk is very useful, if, for example, you would like to read a program with a word processor to edit it. It is assumed that the word processor in this case reads data stored in ASCII code.

This is how the listings in this book were transferred from a Commodore 64 to a Commodore 8032.

In order to print this file on the screen again, you need the following routine:

```
10 OPEN 1,8,2,"TEST.LIST,U,R"
20 GET#1,X$
30 PRINT X$
40 IF ST<>64 THEN 20
50 CLOSE 1
```

This routine is a loop that reads every character (byte) of the file and displays it on the screen. The end of the file is signalled by the status variable which is set to 64 at the end. To send a sequential file to the printer, use the following program:

```
10 OPEN 1,8,2,"TEST.LIST,U,R"
20 OPEN 2,4
30 GET#1,X$
40 PRINT#2,X$
50 IF ST<>64 THEN 30
60 CLOSE 1
```

## Anatomy of the 1541 Disk Drive

Here it assumed that the printer is connected as device address 4.

### 1.4.7 Sequential Files as Tables in the Computer

Sequential data files must reside completely in the computer for data management. Most of the time, a two dimensional table can be used. This table is also called an array or matrix, because a data element can be addressed through the input of two coordinates. To this end, you use a two dimensional variable, which must be reserved with a DIM statement. The first dimension corresponds to the data record, the second dimension to the field inside the record. The following diagram shows an example of a table:

	Field 1	Field 2	Field 3
Record 1	DS(1,1)	DS(1,2)	DS(1,3)
Record 2	DS(2,1)	DS(2,2)	DS(2,3)
Record 3	DS(3,1)	DS(3,2)	DS(3,3)
Record 4	DS(4,1)	DS(4,2)	DS(4,3)
Record 5	DS(5,1)	DS(5,2)	DS(5,3)
Record 6	DS(6,1)	DS(6,2)	DS(6,3)

This table is a file composed of six records which have three fields each. The variable DS is reserved with DIM DS(6,3). To read a sequential file as a table, it is necessary to create such a file with, for example, six records with three fields each. For this purpose, use the following program:

```
100 OPEN 1,8,2,"TABFILE,S,W"
110 FOR X=1 TO 6
120 PRINT CHR$(147)
130 PRINT"RECORD ";X
140 PRINT"-----"
150 FOR Y=1 TO 3
160 PRINT"FIELD ";Y;" : ";
170 INPUT X$
180 PRINT#1,X$
190 NEXT Y
200 NEXT X
210 CLOSE 1
```

Two nested loops are used here, whose variables are numbered with the record and field. Enter six data records. When the program is done, these records will be contained on the

diskette with the filename of **TABFILE**. A tip: save this program with **SAVE"TABPROG",8** so you can use it later.

This file can now be loaded into the computer as a table. Two nested loops indexed for the table are necessary:

```
100 OPEN 1,8,2,"TABFILE.SEO,S,R"
110 DIM D$(6,3)
120 FOR X=1 TO 6
130 FOR Y=1 TO 3
140 INPUT#1,D$(X,Y)
150 NEXT Y
160 NEXT X
170 CLOSE 1
```

This program places data into the table. You can check this with a **PRINT** statements, to see if the data has been stored in the right place. Because each field can be addressed with indices, you can give a command like **PRINT D\$(1,2)** to see the second field of record one. It is meaningful to be able to display the fields of a given record. Use the following routine for this purpose, after you have saved the previous program:

```
100 INPUT"RECORD NUMBER: ";X
110 PRINT"-----"
120 PRINT"FIELD 1: ";D$(X,1)
130 PRINT"FIELD 2: ";D$(X,2)
140 PRINT"FIELD 3: ";D$(X,3)
```

Notice that the first index (the record number) after the question is used as the variable in the field output. The second index (field number) is then constant.

This table can now be altered as desired. Add the following lines to the preceeding program:

```
160 PRINT"-----"
170 INPUT"FIELD TO CHANGE:";Y
180 INPUT"NEW CONTENTS: ";D$(X,Y)
190 PRINT"OK"
200 PRINT"FURTHER CHANGES (Y/N)?"
210 GET X$:IF X$="" THEN 210
220 IF X$="Y" THEN 100
230 IF X$="N" THEN END
240 GOTO 210
```

Here the number of the field to be changed is used as the second index, which is adjacent to the index of the desired record to input the new table element.

This modified table must now be written to the diskette again. You can use the following routine. Don't forget to save the previous edit program first!

## Anatomy of the 1541 Disk Drive

```
100 OPEN 1,8,2,"@:TABFILE,S,W"
110 FOR X=1 TO 6
120 FOR Y=1 TO 3
130 PRINT#1,D$(X,Y)
140 NEXT Y
150 NEXT X
160 CLOSE 1
```

This routine also is relatively short because of the use of nested loops. The @: in line 10 is necessary in order to overwrite the existing file.

Accessing data through the use of the table is very fast. The access time is independent of the size of the table. The size of the table and therefore the quantity of data is dependent on the memory capacity of the computer, however. The large storage area of the Commodore 64 is excellent for table management. If you write a data management program that occupies 8K bytes, then 30K bytes still remain for storing data. If you consider that storing a name and address record of about 80 characters, you can still store 384 records in memory! And this with an access time that cannot be surpassed by refined data management techniques (indexed sequential, relative). But with larger quantities of data, sequential storage is no longer feasible.

### 1.4.8 Searching Tables

As mentioned in the table processing section, each data record of a table can be indexed. Because the table is two dimensional, the first index selects the data record. If a record of the table is to be changed or accessed, the operator must know the record number. The record number can be a part or customer number. There are files, however, for which there is no suitable method of numbering. In such files, the number of the record must be found through a

search of all the records. Here is a practical example:

First of all, create a data file with the following program. Names and telephone numbers are saved in the example:

```
100 OPEN 1,8,2,"TELEDAT,S,W"
110 PRINT CHR$(147)
120 INPUT"LAST NAME      ":";LNS$
130 INPUT"FIRST NAME    ":";FNS$
140 INPUT"AREA CODE     ":";ACS$
150 INPUT"NUMBER        ":";NUS$
160 PRINT"INFORMATION CORRECT (Y/N)?"
170 GETXS:IF XS="" OR XS<>"Y" AND XS<>"N" THEN 170
180 IF XS="N" THEN 110
190 PRINT#1,LNS$,"FNS$","ACS$","NUS$"
```

```

200 PRINT"MORE INPUT (Y/N)?"
210 GETXS:IF XS="" OR XS<>"Y" AND XS<>"N" THEN 200
220 IF XS="N" THEN 240
230 GOTO 110
240 CLOSE 1

```

## Program Documentation:

Line 100	The sequential file "TELEDAT" is opened for writing
Line 110	The screen is cleared
Lines 120-150	The four fields are entered from the keyboard
Lines 160-180	If the data are not correct, they can entered again
Line 190	The four fields are written to disk
Lines 200-220	Here the execution of the program can be ended
Line 230	Input will be continued
Line 240	The file opened in line 100 is closed

Type this program in, RUN it, and enter some data. Save the the program on diskette, so you can combine it with other routines later if you like. In the last section of this chapter, is a complete program for managing your telephone numbers.

If you have entered some data, you would probably like to find a telephone number. To do so, you could print the entire file on the screen or printer and find it yourself. This is, however, a wasteful method, especially if you have entered many records.

The search for the telephone number corresponding to a given name can be performed by the computer. It runs through the whole list, looking for the desired name. Once found, it gives you the complete record which contained that name. The following routine accomplishes this:

```

100 OPEN 1,8,2,"TELEDAT,S,R"
110 DIM D$(100,4):X=1
120 INPUT#1,D$(X,1),D$(X,2),D$(X,3),D$(X,4)
130 IF ST<>64 THEN X=X+1:GOTO 120
140 CLOSE 1
150 PRINT CHR$(147)
160 PRINT"DESIRED NAME: ";NS
170 FOR I=1 TO X
180 ID D$(I,1)=NS THEN 210
190 NEXT I

```



## Anatomy of the 1541 Disk Drive

```
200 PRINT"NAME NOT FOUND!":GOTO 280
210 PRINT"NAME FOUND:"
220 PRINT"-----"
230 PRINT"LAST NAME: ";D$(I,1)
240 PRINT"FIRST NAME: ";D$(I,2)
250 PRINT"AREA CODE: ";D$(I,3)
260 PRINT"NUMBER: ";D$(I,4)
270 PRINT"-----"
280 PRINT"MORE (Y/N)?"
290 GETX$:IF X$="" OR X$<>"Y" AND X$<>"N" THEN 290
300 IF X$="Y" THEN 150
310 PRINT"PROGRAM DONE":END
```

### Program Documentation

Line 100	The sequential file "TELEDAT" is opened for reading
Line 110	The table is dimensioned for 100 records and the index is set to one
Line 120	The data records are read into the table
Line 130	The status variable ST is checked for end of file (indicated by a value of 64). If the end has not been reached, the index is incremented and a new record is read.
Line 140	The file opened in line 100 is closed
Line 150	The screen is cleared
Line 160	The last name to be searched for is read from the keyboard and placed in the variable N\$
Lines 170-190	The loop searches the table of records, checking the name fields against the desired name. If the position is found, the program branches to the output routine
Line 200	The name was not found
Lines 210-270	The record containing the desired name is displayed
Lines 280-310	The possibility to search for a new name is allowed

You will notice that this search is quite fast when the data is already loaded into the computer. Searching the computer's memory is faster than searching the diskette. The program can be easily changed to search for a desired field other than the name. You might want to search for an area code, for instance. The first program stops the search when the first matching data record is found. This is not always

desired, however. If, for instance, you wish to search the table looking for a particular area code and want all matches to be displayed, a different routine is needed. The routine must continue the search after the first match is found. The next program takes care of this:

```

100 OPEN 1,8,2,"TELEDAT,S,R"
110 DIM D$(100,4):X=1
120 INPUT#1,D$(X,1),D$(X,2),D$(X,3),D$(X,4)
130 IF ST<>64 THEN X=X+1:GOTO 120
140 CLOSE 1
150 PRINT CHR$(147)
160 PRINT"AREA CODE TO SEARCH FOR: ";AC$
170 FOR I=1 TO X
180 IF D$(I,3)=AC$ THEN 210
190 NEXT I
200 PRINT"END OF DATA!":GOTO 270
210 PRINT"-----"
220 PRINT"LAST NAME:      ";D$(I,1)
230 PRINT"FIRST NAME:     ";D$(I,2)
240 PRINT"AREA CODE:      ";D$(I,3)
250 PRINT"NUMBER:         ";D$(I,4)
260 PRINT"-----"
270 PRINT"MORE (Y/N)?"
280 GETX$:IF X$="" OR X$<>"Y" AND X$<>"N" THEN 280
290 IF X$="Y" THEN 190
300 PRINT"SEARCH DONE!":END

```

Here the search is continued if a record with the appropriate area code is found. This happens in line 290, which branches back to the loop instead of ending the program. After searching all of the records, the program responds **END OF DATA**. If you understand the operation of this program, you can now develop a search for the last name. With the help of the previous programs, this should present no difficulty.

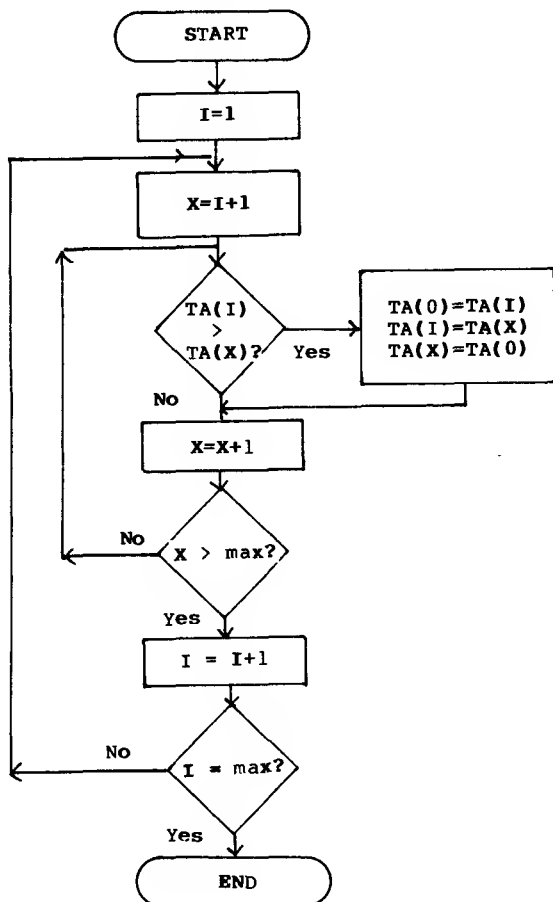
#### 1.4.9 Simple Sorting of Tables

In data processing, it is often necessary to sort data into numeric or alphabetic order. This has always been a time consuming task, which the programmer has tried to shorten by using better sorting methods. Sorting is certainly a time consuming task when performed with the programming language BASIC, which is relatively slow.

Why should we sort the data at all? Suppose you had a telephone book in which the names were not ordered. You would have search the entire book from beginning to end to find a name. Sorting offers advantages when searching data. The computer can also search sorted data faster.

## Anatomy of the 1541 Disk Drive

There are several search methods which differ mainly in their speed of execution. The simplest method compares each data item with every other. If a table is supposed to be sorted in ascending order, the first item in the table is compared to the second. If the first is greater, it is exchanged with the second. After that, the first will be compared to the third, and so on, until the last item is reached. Now the smallest item is at the beginning, in the right place. The next time through, the first item is no longer needed. A flowchart of the program logic appears below.



This sort program starts using an index of 1, which is stored in the variable I. The second index is the variable X, which receives a value one greater than I. Then the first item is compared to the second. If the value of TA(I) is greater than TA(X), the program must use a temporary variable, TA(0), to make the exchange between the two. After this, the value of X is incremented, to three, and TA(I) is again compared to TA(X), etc. When the last item in the table is reached, (X > last index), the first item will be the smallest, and the index I is incremented by one. Now the second item is compared to every other (starting with the third), and so on.

This sort method looks quite complicated at first glance. Comparisons in memory are done relatively quickly, however. This method is sufficient for small quantities of data.

In order to run this program, a table must be built. This example uses a table with twelve items containing alphanumeric data (strings). The table is filled by the following routine:

```
100 DIM TAS(12)
110 FOR I=1 TO 12
120 INPUT TAS(I)
130 NEXT I
```

This program allows you to enter twelve strings, which are then sorted with the following program:

```
140 I=1
150 X=I+1
160 IF TAS(I) < TAS(X) THEN 180
170 TAS(0)=TAS(I):TAS(I)=TAS(X):TAS(X)=TAS(0)
180 X=X+1
190 IF X <= 12 THEN 160
200 I=I+1
210 IF I <> 12 THEN 150
220 FOR I=1 TO 12
230 PRINT TAS(I)
240 NEXT I
```

The table is sorted and displayed on the screen. If, instead of a one dimensional table, you want to sort a two dimensional table such as our telephone file, exchange the fields by changing lines 160-170 as below:

```
160 IF DS(I,1) < DS(X,1) THEN 180
170 DS(0,1)=DS(I,1):DS(I,1)=DS(X,1):
  DS(X,1)=DS(0,1)
171 DS(0,2)=DS(I,2):DS(I,2)=DS(X,2):
  DS(X,2)=DS(0,2)
172 DS(0,3)=DS(I,3):DS(I,3)=DS(X,3):
  DS(X,3)=DS(0,3)
173 DS(0,4)=DS(I,4):DS(I,4)=DS(X,4):
  DS(X,4)=DS(0,4)
```

## Anatomy of the 1541 Disk Drive

It is very time consuming to sort a greater amount of data with this method. If you have a large amount of data to be sorted, we recommend that you use the very fast machine language sort routine from our book *Commodore 64 Tips & Tricks*.

### 1.4.10 MAILING LIST MANAGEMENT with Sequential Data Storage

At the end of this section, is a mailing list management program that every user will hopefully find easy to use. At the same time, this program provides insight into the operation of many data processing techniques.

A mailing list record of this program consists of the following fields:

- NAME 1
- NAME 2
- STREET
- CITY, STATE
- ZIP CODE
- TELEPHONE NUMBER
- NOTES

The use of the fields 'NAME 1' and 'NAME 2' are up to the user. For instance, 'NAME 1' can be the first name and 'NAME 2' the last name, or 'NAME 1' the company name and "to the attention of..." in 'NAME 2'. The field 'NOTES' can be used for grouping the addresses (family, business, friends, etc.).

The program offers the following Main Menu options:

- 1- LOAD DATA
- 2- SAVE DATA
- 3- INPUT DATA
- 4- EDIT DATA
- 5- SELECT/PRINT DATA
- 6- DELETE DATA
- 0- END PROGRAM

#### -1- LOAD DATA

Use this function to enter the name of the mailing list file that is to be maintained. If the file exists on the diskette, it is loaded and ready to be used. The number of records in the file is displayed. If an error is encountered while loading, or if the file does not exist, the message **DISK ERROR!** is displayed. At the conclusion of this function, the Main Menu reappears.

## -2- SAVE DATA

Use this function to write an updated or expanded copy of the mailing list to the diskette. If the file name already exists, then the file is overwritten.

The mailing list should be saved often while using the program in case a power outage should erase the computer's memory. After saving, the file can be used further, without having to reload it in again.

## -3- INPUT DATA

Use this function to add records to the mailing list:

1. When no data has been previously loaded.

First a file name for the mailing list is entered. Enter a file name which does not already exist on the diskette or the old file is overwritten. All records that are inputted are new to the mailing list.

2. When data has been previously loaded.

All records that are inputted are added to the existing mailing list.

After entering an mailing list entry, the message **CORRECT (Y/N)?** is displayed. Here you may correct the data. If the entry is not correct, press the **N** key. If the entry is correct, press **Y**. Now the message **MORE INPUT (Y/N)?** is displayed. If you want to enter another mailing list entry, press **Y**. If you press **N**, the Main Menu appears again.

## -4- EDIT DATA

Use this function to change existing mailing list records. Both Name 1 and Name 2 must be entered. If both names are not known, the other can be found with the **SELECT/PRINT DATA** routine. After entering the names, the mailing list is searched for matching names. When they are found, the complete address is displayed with the fields numbered. Now you must enter the number of the field which you want to change. The new contents are requested. The record is once again displayed in its updated form. If no more changes to this record are required, press **9**. The program asks if another record is to be changed. This question is to be answered by pressing **Y** or **N**.

## Anatomy of the 1541 Disk Drive

### -5- SELECT/PRINT DATA

Use this function to search for certain records and print or display them. You must first specify if the selected records are to be printed on the screen (S) or the printer (P). If you have selected the printer, you must again choose if the data is to be printed with all fields on normal paper (P), or if fields 1-5 are to be printed on mailing labels (M). The address labels must be in a single column and measure 89mm x 36mm.

In order to select the data, enter search criteria. For fields which are not relevant, simply press RETURN. If, for example, you want to find all addresses in Grand Rapids, press RETURN for the first three fields and type **GRAND RAPIDS, MI** for the fourth, and press RETURN for the next three.

An example:

```
NAME 1           : M
NAME 2           : <return>
STREET          : <return>
CITY, STATE     : <return>
ZIP CODE        : <return>
TELEPHONE NUMBER : <return>
NOTES           : FAMILY
```

All family members whose name 1 begins with 'M' will be displayed.

You can see how versatile this search is. Try it out yourself.

### -6- DELETE DATA

Use this function to delete records. After entering the first and second names of the record, the record is read and the remaining fields are displayed. Then you are asked to confirm that the record is to be deleted. If you press Y, the record is deleted.

### -0- END PROGRAM

Use this function to leave the program. Before the program is ended, you are reminded that you can restart the program without losing data by typing **GOTO 110**. This is important if you forget to save the data before ending the program.

Here is the program listing:

```

100 POKE 53280,5:POKE53281,2:PRINTCHR$(158);:DIMD$(100,7)
110 GOSUB2030
120 PRINT"SELECT THE DESIRED FUNCTION:"
130 PRINT"-----":PRINT
140 PRINT"      -1- LOAD DATA"
150 PRINT"      -2- SAVE DATA"
160 PRINT"      -3- INPUT DATA"
170 PRINT"      -4- EDIT DATA"
180 PRINT"      -5- SELECT/PRINT DATA"
190 PRINT"      -6- DELETE DATA":PRINT
200 PRINT"      -0- END PROGRAM"
210 PRINT
220 PRINT"          CHOICE (0-6)?"
230 GETX$:IFX$<"0"ORX$>"6"THEN230
240 IF X$<>"0"THEN340
250 PRINT:PRINT"          ARE YOU SURE (Y/N)?"
260 GETX$:IFX$<>"N"ANDX$<>"Y"THEN260
270 IFX$="N"THEN110
280 GOSUB2030
290 PRINT"THE PROGRAM CAN BE RESTARTED WITH
300 PRINT"          'GOTO 110'"
310 PRINT"          WITHOUT LOSS OF DATA"
330 END
340 ONVAL(X$)GOSUB360,540,680,880,1190,1770
350 GOTO 110
360 REM *****
370 REM LOAD DATA
380 REM *****
390 GOSUB 2030
400 INPUT"NAME THE FILE :";FNS$
410 OPEN 15,8,15
420 OPEN1,8,2,FNS$+",S,R"
430 INPUT#15,FE:IF FE=0 THEN 460
440 PRINT"DISK ERROR!"
450 GOTO 510
460 X=1
470 INPUT#1,D$(X,1),D$(X,2),D$(X,3),D$(X,4),D$(X,5),D$(X,6),
    D$(X,7)
480 IF ST<>64 THEN X=X+1:GOTO470
490 PRINT"FILE IS LOADED AND CONTAINS";X;"RECORDS."
500 PRINT
510 CLOSE:CLOSE15
520 PRINT"RETURN FOR MORE"
530 INPUTX$:RETURN
540 REM *****
550 REM SAVE DATA
560 REM *****
570 IF X>0 THEN 590
580 GOSUB2230:RETURN
590 GOSUB 2030
600 OPEN 1,8,2,"@:"+FNS$+",S,W"
610 FORI=1TOX
620 PRINT#1,D$(I,1)","D$(I,2)","D$(I,3);

```



# Anatomy of the 1541 Disk Drive

```

630 PRINT#1,D$(1,4)","D$(1,5)","D$(1,6)","D$(1,7)
640 NEXT
650 PRINT"DATA IS SAVED":CLOSE1:RETURN
660 PRINT"RETURN FOR MORE"
670 INPUTX$:RETURN
680 REM *****
690 REM INPUT DATA
700 REM *****
710 IFX>0THEN730
720 GOSUB2030:INPUT"FILENAME ";FN$
730 X=X+1
740 GOSUB2030
750 PRINT"INPUT DATA:"
760 PRINT"-----":PRINT
770 I=X:GOSUB2110
780 FORI=1TO7:PRINTCHR$(145);:NEXT
790 FORI=1TO7:PRINTTAB(12);:INPUTD$(X,I):NEXT
800 PRINT:PRINT"CORRECT (Y/N)?"
810 GETX$:IFX$<>"N"ANDX$<>"Y"THEN810
820 IFX$="Y"THEN840
830 GOTO 740
840 PRINT"MORE INPUT (Y/N)?"
850 GETX$:IFX$<>"N"ANDX$<>"Y"THEN850
860 IFX$="Y"THEN730
870 RETURN
880 REM *****
890 REM EDIT DATA
900 REM *****
910 IF X>0THEN930
920 GOSUB2230:RETURN
930 GOSUB2030
940 INPUT"NAME 1: ";N1$
950 INPUT"NAME 2: ";N2$
960 FORI=1TOX
970 IF D$(I,1)=N1$ANDD$(I,2)=N2$THEN1010
980 NEXTI
990 PRINT"NAME NOT FOUND!"
1000 PRINT"RETURN FOR MORE":INPUTX$:RETURN
1010 GOSUB2030
1020 PRINT"-1- NAME 1 :";D$(I,1)
1030 PRINT"-2- NAME 2 :";D$(I,2)
1040 PRINT"-3- STREET :";D$(I,3)
1050 PRINT"-4- CITY, STATE :";D$(I,4)
1060 PRINT"-5- ZIP CODE :";D$(I,5)
1070 PRINT"-6- TELEPHONE :";D$(I,6)
1080 PRINT"-7- NOTES :";D$(I,7)
1090 PRINT"NO. OF FIELD TO CHANGE: ":PRINT"(9=NO
CHANGES)"
1100 GETX$:IFVAL(X$)<1ORVAL(X$)>7ANDVAL(X$)<>9THEN1100
1110 IFVAL(X$)=9THEN1150
1120 Y=VAL(X$)
1130 INPUT"NEW CONTENTS";D$(I,Y):PRINT
1140 GOTO 1010
1150 PRINT"MORE CHANGES (Y/N)?"
1160 GETX$:IFX$<>"Y"ANDX$<>"N"THEN1160

```

```

1170 IFX$="Y"THEN880
1180 RETURN
1190 REM *****
1200 REM SELECT/PRINT DATA
1210 REM *****
1220 IF X>0THEN1240
1230 GOSUB2230:RETURN
1240 GOSUB2030:PRINT"OUTPUT TO PRINTER (P) OR SCREEN (S)?"
1250 GETX$:IFX$<>"S"ANDX$<>"P"THEN1250
1260 O$=X$:IFO$="S"THEN1300
1270 PRINT:PRINT"PAPER (P) OR MAILING LABELS (M)?"
1280 GETX$:IFX$<>"P"ANDX$<>"M"THEN1280
1290 D$=X$
1300 GOSUB2030
1310 PRINT"ENTER THE SEARCH DATA:"
1320 PRINT"PRESS RETURN BY IRRELEVANT FIELDS."
1330 PRINT"-----":PRINT
1340 I=0:GOSUB2110
1350 FORI=1TO7:PRINTCHR$(145);:S$(I)="":NEXT
1360 FORI=1TO7:PRINTTAB(12);:INPUTS$(I):NEXT
1370 IFO$="S"ORD$="M"THEN1450
1380 GOSUB2030:PRINT"PRINTER READY (Y)?"
1390 GETX$:IFX$<>"Y"THEN1390
1400 OPEN 1,4
1410 PRINT#1,"NAME 1";SPC(8);"NAME 2";SPC(8);"STREET";
    SPC(10);
1420 PRINT#1,"CITY, STATE";SPC(4);"ZIP CODE TELEPHONE NOTES"
1430 FORI=1TO79:PRINT#1,"=";:NEXT:PRINT#1
1440 CLOSE1
1450 FORI=1TOX
1460 FORY=1TOY
1470 IFSS(Y)=LEFT$(D$(I,Y),LEN(SS(Y)))THENZ=Z+1:GOTO1480
1480 NEXTY
1490 IFZ=7THENGOSUB1550
1500 Z=0:NEXTI
1510 PRINT:PRINT"END OF DATA!":PRINT
1520 PRINT"RETURN FOR MORE":PRINT
1530 INPUTX$
1540 RETURN
1550 IFO$="S"THEN1730
1560 IFD$="M"THEN1670
1570 OPEN1,4
1580 PRINT#1,D$(I,1);SPC(14-LEN(D$(I,1)));
1590 PRINT#1,D$(I,2);SPC(14-LEN(D$(I,2)));
1600 PRINT#1,D$(I,3);SPC(16-LEN(D$(I,3)));
1610 PRINT#1,D$(I,4);SPC(15-LEN(D$(I,4)));
1620 PRINT#1,D$(I,5);SPC(8-LEN(D$(I,5)));
1630 PRINT#1,D$(I,6);SPC(12-LEN(D$(I,6)));
1640 PRINT#1,D$(I,7)
1650 PRINT#1:CLOSE1
1660 RETURN
1670 OPEN2,4
1680 PRINT#2
1690 FORJ=1TO5:PRINT#2,D$(I,J):NEXT
1700 PRINT#2:PRINT#2:PRINT#2

```

# Anatomy of the 1541 Disk Drive

```

1710 CLOSE2
1720 RETURN
1730 GOSUB2030:GOSUB2110
1740 PRINT:PRINT"MORE (Y)?"
1750 GETXS:IFX$<>"Y"THEN1750
1760 RETURN
1770 REM *****
1780 REM DELETE DATA
1790 REM *****
1800 IFX>0THEN1820
1810 GOSUB2230:RETURN
1820 GOSUB2030
1830 INPUT"NAME 1 : ";N1$
1840 INPUT"NAME 2 : ";N2$
1850 FORI=1TOX
1860 IFD$(I,1)=N1$ANDD$(I,2)=N2$THEN1900
1870 NEXTI
1880 PRINT"NAME NOT FOUND!":PRINT
1890 PRINT"RETURN FOR MORE":INPUTXS:RETURN
1900 GOSUB2030:GOSUB2110
1910 PRINT:PRINT"DELETE RECORD (Y/N)?"
1920 GETXS:IFX$<>"Y"ANDXS<>"N"THEN1920
1930 IFX$="N"THENRETURN
1940 FORY=ITOX-1
1950 FORJ=1TO6
1960 D$(Y,J)=D$(Y+1,J)
1970 NEXTJ,Y
1980 FORJ=1TO6:D$(X,J)="":NEXTJ
1990 X=X-1
2000 PRINT"RECORD IS DELETED!"
2010 PRINT"RETURN FOR MORE"
2020 INPUTXS:RETURN
2030 REM *****
2040 REM PROGRAM HEADING
2050 REM *****
2060 PRINTCHR$(147);
2070 PRINTTAB(8);"=====
2080 PRINTTAB(8);"M A I L I N G   L I S T
2090 PRINTTAB(8);"=====
2100 RETURN
2110 REM *****
2120 REM PRINT RECORD
2130 REM *****
2140 PRINT"NAME 1      : ";D$(I,1)
2150 PRINT"NAME 2      : ";D$(I,2)
2160 PRINT"STREET      : ";D$(I,3)
2170 PRINT"CITY, STATE : ";D$(I,4)
2180 PRINT"ZIP CODE    : ";D$(I,5)
2190 PRINT"TELEPHONE   : ";D$(I,6)
2200 PRINT"NOTES       : ";D$(I,7)
2220 RETURN
2230 REM *****
2240 REM NO DATA!
2250 REM *****
2260 GOSUB2030

```

```
2270 PRINT"NO DATA IN MEMORY!":PRINT
2280 PRINT"RETURN FOR MORE"
2290 INPUTXS:RETURN
```

#### 1.4.11 Uses for Sequential Storage

The great advantage of sequential storage as compared to relative and direct access storage, is that a lot of data can be written to the diskette quickly. Data of varying lengths can be stored together, without requiring the records to be of a definite length. It makes sense to make use of this advantage, where the file must not be permanently divided into parts. Examples are:

- \* Bookkeeping files

In a bookkeeping journal, all entries are recorded continuously. Changes should not be made to these entries. Instead, adjustment entries should be made to effect changes.

- \* Analysis files

You analyze a direct access file, looking for, say, all customers with whom you have done more than 2000 dollars of business in a certain zip code, and write the found records in a sequential file for later access.

Naturally, sequential files also offer a substitute for direct access files, as discussed in this chapter, if the user does not possess further programming knowledge. We must certainly recommend that you work through the other methods of data storage, which offer other advantages.

### 1.5 Relative Data Storage

Relative data storage and its programming is not described in the VIC-1541 user's manual. The reason may lie in the fact that the Commodore 64 and the VIC-20 have no commands to process relative files using BASIC 2.0. Therefore, it is in principle not possible to use relative data storage on the Commodore 64 and VIC-20 - but only in principle. We have developed a few tricks that work within the limitations of BASIC 2.0 and permit the Commodore 64 and also the VIC-20 to use relative data storage. The examples may seem to be somewhat complicated at first. For example, information about the record lengths will be transmitted to the disk using CHR\$(x) codes. But they provide for a very easy method of data storage.

#### 1.5.1 The Principle

When using relative record data processing, the data records are numbered. It is assumed that all records in a relative file have the same length and that the record number of every record is known or can be calculated. To find a record, it is not necessary to search through the entire file. Only the record number need be given to access the record. Using the record number, the DOS can find where the record is "relative" to the beginning of the file on the diskette and can read it directly. Therefore, you don't have to read an entire file into the computer, only the desired records.

Managing a relative file follows this pattern:

Create a relative file:

1. The file is opened. With this the length of a record is established.
2. The last record is marked.
3. The file is closed.

Writing a record:

1. The file is opened.
2. The file is positioned on the record to be written.
3. The record is written.
4. The file is closed.

Reading a record:

1. The file is opened.
2. The file is positioned over the record to be read.
3. The record is read.
4. The file is closed.

This is only an outline. In the following sections these processes will be explained in detail.

## 1.5.2 The Advantage over Sequential Storage

The greatest advantages of relative storage are:

- \* faster access to individual records
- \* does not require much of the computer's memory

It has already been mentioned that the sequential file must reside completely in the computer's memory for processing. Using sequential techniques, it may be necessary to search the entire file to find a given record. The record must be read and compared during the search process. But if a sequential file cannot be entirely loaded into memory, this method of search is impossible.

Using relative data files, the processing is much simpler. By using the record number, a desired record can be read individually. The file size is not limited to the computer's memory. So, for example, a program that uses all 3.5K bytes of a standard VIC-20 can manage a file with up to 163 Kbytes!

The advantages of relative over sequential file management are large enough that many of you, once acquainted with the techniques will prefer to use them.

## 1.5.3 Opening a Relative File

Relative files are also opened with the OPEN command. The command differs only slightly from that for sequential files. Take a look at the format of the OPEN command:

**OPEN lfn,da,channel,"filename,L,"+CHR\$(recordlength)**

The first four parameters are identical to those for sequential files. They are logical file number, device address (normally 8), channel (2-14), and name of the file. Next follows an L which informs the DOS that a relative file should be opened, whose record length follows. This record length is transmitted with a CHR\$ code. The length is between one and 254. Thus each record of a relative file is limited to a maximum of 254 characters.

If the record length is smaller than 88, the record can be read with an INPUT# statement. For this, it is necessary

## Anatomy of the 1541 Disk Drive

that the PRINT# statement transfers the record with a trailing RETURN. A PRINT# statement sends a RETURN when it is not ended with a semicolon. This RETURN is now a part of the record. When you want to read records with INPUT#, the record length must be increased by one.

A file composed of 80-character records, to be read by the INPUT# statement would be opened as follows:

```
OPEN 1,8,2,"FILE.REL,L,"+CHR$(81)
```

Here a relative file with the name "FILE.REL" is opened using channel 2. The record length should total 81 characters. Records comprised of 80 characters should be sent with a PRINT# statement, with no trailing semicolon.

It is important to note that only one relative file can be opened at a time. If you want to work with two relative files, you must always close the first before opening the second. One sequential file may be opened in addition to one relative file.

When a relative file is opened for the first time, the DOS creates as many "null" or unused records that can fit in a single 254 byte block. It creates these "null" records by writing a record with a CHR\$(255) at the beginning of each record. This is called formatting a relative file.

If you want to expand a relative file beyond the initial number of records that the DOS formatted, then you can reference the last record number that you want to write (by positioning to that record number) and the DOS automatically formats the records between the current end of file and the new last record number by writing records containing CHR\$(255). Formatting takes time to complete.

If you try to read a record whose number greater than that of the last record, the DOS returns the error **RECORD NOT PRESENT**. However, if you write a record which is greater than the highest current record, all records less than the new record number are also written with CHR\$(255). Subsequently accessing these record does not result in an error.

If you want to avoid long delays as relative records are formatted (as the file is expanded), then you should reference the last record number immediately after opening the file. The formatting of the null records takes place at that time instead of at a more inconvenient time.

To position the DOS for a specific relative record you must send a position command over the command channel (15), as shown here:

```
PRINT#1fn,"P"+CHR$(channel)+CHR$(low)+CHR$(high)+CHR$(byte)
```

If you are positioning to a record which is beyond the current end of file, the DOS presents the message **RECORD NOT PRESENT** appears to the disk error channel. If this record is to be written, then you can ignore the message. The following **PRINT#** statement is carried out in spite of the error message.

The parameters **low** and **high** in the **P** command designate the record number. The maximum value that can be given with one byte is 255, but a relative file contains up to 65535 records. Therefore, the record number must be transmitted in two bytes. These two bytes are calculated with the following formula:

$$\begin{aligned} \text{HB} &= \text{INT}(\text{RN}/256) \\ \text{LB} &= \text{RN} - \text{HB} * 256 \end{aligned}$$

HB = High Byte (parameter high)  
LB = Low Byte (parameter low)  
RN = Record Number

The last parameter (byte) serves to position to a specific location within the given record. An example:

**PRINT#2,"P"+CHR\$(2)+CHR\$(10)+CHR\$(1)+CHR\$(5)**

Here the file is positioned to the fifth byte of the 266th record. This 266 is coded as a low byte of 10 and a high byte of 1 (high byte \* 256 + low byte = record number).

To read or write a complete record, the file is positioned to the first byte of the record. If the last parameter is not given, the trailing **RETURN** (**CHR\$(13)**) is taken as the character location.

The corresponding BASIC program to establish a file of 100 80-character records looks like this:

```
100 RN=100
110 HB=INT(RN/256)
120 LB=RN-HB*256
130 OPEN1,8,2,"FILE.REL,L,"+CHR$(80)
140 OPEN2,8,15
150 PRINT#2,"P"+CHR$(2)+CHR$(LB)+CHR$(HB)+CHR$(1)
160 PRINT#1,CHR$(255)
170 CLOSE 1:CLOSE 15
```

Freeing 100 records takes some time. The creation of this file takes about ten minutes. Notice that of the 80 characters in a record, only 79 can be used to hold data, because transferring data with a **PRINT#** command adds a trailing **RETURN**.



## 1.5.4 Preparing Data for Relative Storage

As already mentioned, you cannot change the record length of a relative file. If a record consists of several fields, these fields must be combined. It is important that these fields always be in the same position so that they can be separated later. Let's work through a problem:

We want to manage an inventory using relative storage techniques. To that end, the following fields are necessary:

```

PART NUMBER      4 CHARACTERS
DESCRIPTION      15 CHARACTERS
QUANTITY         5 CHARACTERS
COST             6 CHARACTERS
PRICE           6 CHARACTERS
-----
Record length 36 bytes
=====

```

The inventory contains approximately 200 items with a record length of 36 bytes. This inventory file can now be created:

```

100 RN=200:REM NUMBER OF INVENTORY ITEMS
110 RL=36 :REM RECORD LENGTH
120 OPEN 1,8,2,"INVEN,L,"+CHR$(36)
130 OPEN 2,8,15
140 PRINT#2,"P"+CHR$(2)+CHR$(200)+CHR$(0)+CHR$(1)
150 PRINT#1,CHR$(255)
160 CLOSE 1:CLOSE 2

```

Now the file is created and all records are written. Let's suppose that the inventory is present as a sequential file. It consists of 200 records, the fields of which are ordered one after the other. These fields must be written to the relative file. This is not simple, however, because many of the descriptions are not the full fifteen characters in length, for example. The structure of the relative file looks as follows:

```

11111111111222222222223333333
Position : 123456789012345678901234567890123456
=====
Field    : P$-D$-----Q$---C$----P$----
=====
Contents : 1    1/8 in. sheet 1344 11.40 20.30
           : 2    No. 10 screw 1231  4.00  7.00
           : 3    Valve A3A4   1243 11.45 16.40
           :      .
           :      .
           :      .
           : 200 1/2 in. tubing 2321  3.35  4.10

```

The fields will be read from the sequential file into the following variables:

Part number	PN\$
Description	DE\$
Quantity	Q\$
Cost	C\$
Price	P\$

The following command chains these fields together:

```
RC$ = PN$ + DE$ + Q$ + C$ + P$
```

The record variable RC\$ does not have the desired structure. The reason is that the quantity immediately follows the description. Because the quantity must begin at position 20 and the description is not always fifteen characters, we have a problem. In order to read the records from the relative file, the structure must be observed. Therefore, all fields that are shorter than the planned length must be padded with blanks. Taking this into account, the chaining goes like this:

```
BL$="
RC$=PN$+LEFT$(BL$,4-LEN(PN$))
RC$=RC$+DE$+LEFT$(BL$,15-LEN(DE$))
RC$=RC$+Q$+LEFT$(BL$,5-LEN(Q$))
RC$=RC$+C$+LEFT$(BL$,6-LEN(C$))
RC$=RC$+P$+LEFT$(BL$,6-LEN(P$))
```

This concatenation looks more complicated than it really is. Each field must be filled with enough blanks to bring it to its appropriate length. The blanks are added to the individual fields from the string BL\$, defined at the beginning. T

Let's go through an example:

Suppose the first part number is 8. The length of this string, LEN(PN\$), is then one. The maximum length of this field (4) minus the actual length (1) is 3. The string PN\$ must therefore be padded with three blanks, LEFT\$(BL\$,3).

Each record of the old sequential file must be prepared in this manner before it can be transferred to the relative file.

Naturally, the above is true for all input values to be used in a relative file. Therefore, you must always remember to use a routine to fill each field with blanks to its full length when working with relative data processing.

## 1.5.5 Transferring Data

In principle, transferring data to and from a relative file does not differ from sequential storage. Records are written with PRINT# and read with INPUT# or GET#. The only difference is that before a record is be written or read, the file must be positioned to that record. This is accomplished with the P command. This example program illustrates what we have discussed:

```

100 BL$="
105 OPEN 1,8,2,"TEST.REL,L,"+CHR$(41)
110 OPEN 2,8,15
120 PRINT#2,"P"+CHR$(2)+CHR$(100)+CHR$(0)+CHR$(1)
130 PRINT#1,CHR$(255)
140 PRINT CHR$(147)
150 PRINT"INPUT RECORD:"
160 PRINT"-----"
170 INPUT"RECORD NUMBER (1-100)  : ";RN
180 IF RN<1 OR RN>100 THEN PRINTCHR$(145);:GOTO160
190 INPUT"FIELD 1 (MAX.10 CHAR.) : ";F1$
200 IF LEN(F1$)>10 THEN PRINTCHR$(145);:GOTO190
210 INPUT"FIELD 2 (MAX. 5 CHAR.) : ";F2$
220 IF LEN(F2$)>5 THEN PRINTCHR$(145);:GOTO210
230 INPUT"FIELD 3 (MAX.10 CHAR.) : ";F3$
240 IF LEN(F3$)>10 THEN PRINTCHR$(145);:GOTO230
250 INPUT"FIELD 4 (MAX.15 CHAR.) : ";F4$
260 IF LEN(F4$)>15 THEN PRINTCHR$(145);:GOTO250
270 PRINT"CORRECT (Y/N)?"
280 GETX$:IF X$<>"Y" AND X$<>"N" THEN 280
290 IF X$="N" THEN 140
300 RC$=F1$+LEFT$(BL$,10-LEN(F1$))
310 RC$=RC$+F2$+LEFT$(BL$,5-LEN(F2$))
320 RC$=RC$+F3$+LEFT$(BL$,10-LEN(F3$))
330 RC$=RC$+F4$+LEFT$(BL$,15-LEN(F4$))
340 PRINT#2,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
350 PRINT#1,RC$
360 PRINT"MORE INPUT (Y/N)?"
370 GETX$:IF X$<>"Y" AND X$<>"N" THEN 370
380 IF X$="Y" THEN 140
390 CLOSE 1:CLOSE 2:END

```

The following line-oriented documentation explains the operation of the program:

```

100      A blank-character string with 15 blanks is
        defined.
105      The relative file is opened with a length of 15.
110      The command channel 15 is opened.
120      To initialize the relative file, the head is
        positioned over the first byte of the last (100th)
        record.
130      The last record is freed and the initialization
        begun.
140      The screen is erased.

```

150-260	The record no. and fields 1-4 are entered and checked for correct length.
270-290	The entered data can be corrected.
300-330	The record is prepared.
340	The head is positioned over the first byte of the record.
350	The record is written to the disk.
360-380	New data can be entered.
390	The program ends.

Now write some records with this program, but don't forget to save in case you need it later.

Certainly, it also necessary to read and change existing records. To do this, the relative file is opened, the file is positioned to the appropriate record, and the record is read. This record must then be divided into its fields. Let's read a record that was recorded with the previous program. The following routine reads the record:

```

100 OPEN 1,8,2,"TEST.REL,L,"+CHR$(41)
110 OPEN 2,8,15
115 PRINT CHR$(147)
120 INPUT"RECORD NUMBER :";RN
130 PRINT#2,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
140 INPUT#1,RC$
160 IF ASC(RC$)<>255 THEN PRINT"RECORD NOT FOUND!":
    GOTO250
170 PRINT RC$
250 CLOSE 1:CLOSE 2

```

This routine reads a specified record. If this record has never been written, it is recognized by the value 255 with which every record was marked at the establishment of the file.

A record that is found is displayed. You can see that the four fields are in the same positions. If you want to divide the record into its individual parts, you must use the function MID\$. For example, in order to extract field 1 of the record, give the following statements in the direct mode after the record is found and read:

```

F1$=MID$(RC$,1,10)
PRINT F1$

```

Now the variable F1\$ contains the first field, as written by the first program. The division of records into individual fields is accomplished by building on the previous program. Add or change the following lines:

```

170 F1$=MID$(RC$,1,10)
180 F2$=MID$(RC$,11,5)
190 F3$=MID$(RC$,16,10)
200 F4$=MID$(RC$,26,15)

```

## Anatomy of the 1541 Disk Drive

```
210 PRINT"FIELD 1: ";F1$
220 PRINT"FIELD 2: ";F2$
230 PRINT"FIELD 3: ";F3$
240 PRINT"FIELD 4: ";F4$
250 PRINT"MORE (Y/N)?"
260 GETX$:IF X$<>"Y" AND X$<>"N" THEN 260
270 IF X$="Y" THEN 115
280 CLOSE 1:CLOSE 2
```

Here the record is separated into the individual fields and the fields are displayed. It is important for the MID\$ function that the exact positions of the fields within the record be maintained. The first parameter within the parentheses is the string variable containing the record. The second parameter is the position at which the number of characters represented by the parameter will be taken out. Further work may be done with the selected fields inside the program.

So far, we have read the records with the INPUT# statement. If the record is longer than 88 characters, it can no longer be read with the INPUT# statement. The way to get around the limited INPUT# statement is with the GET# statement. The bytes of a record are read one at a time with this command and assembled into a single string. Suppose you have a relative file with 128-character records. Now you want to read the tenth record of this file and place it in the variable RC\$. The example of the following routine illustrates reading this with GET#:

```
100 OPEN 1,8,2,"TEST.GET,L,"+CHR$(128)
110 OPEN 2,8,15
120 PRINT#2,"P"+CHR$(2)+CHR$(10)+CHR$(0)+CHR$(1)
130 RC$=""
140 FOR I=1 TO 128
150 GET#1,X$
160 RC$=RC$+X$
170 NEXT I
.
.
.
.
```

After running this routine, the record is contained in the variable RC\$. If this record had been written with a PRINT# statement without a trailing semicolon, the last character in the string will be a RETURN. To ignore this RETURN, allow the loop in line 140 to run only to 127. The last character of the record RETURN is not read.

As already mentioned, the last parameter of the P command specifies at which character the transfer of data should begin. If, for instance, in the 127-character record of the previous example, you want to read positions 40-60 into a

field, the head must be positioned over the 40th character and the next 21 bytes read. The following routine clarifies this:

```

100 OPEN 1,8,2,"TEST.GET,L,"+CHR$(128)
110 OPEN 2,8,15
120 PRINT#2,"P"+CHR$(2)+CHR$(10)+CHR$(0)+CHR$(40)
130 F$=""
140 FOR I=1 TO 21
150 GET#1,X$
160 F$=F$+X$
170 NEXT I
.
.
.
.

```

In line 120, the head is positioned over the the 40th byte of the tenth record in line 120 and the loop in lines 140-170 reads the following 21 bytes (bytes 40-60 of the record) into F\$.

You see then that the entire record need not be read if you only want to work with part of it.

## 1.5.6 Closing a Relative File

There is no difference between closing a relative file and sequential file. Because the command channel must always be open to send the position command when working with relative storage, it must also be closed.

## 1.5.7 Searching Records with the Binary Method

Normally each record is accessed by record number. But what if you want to search for a specific name in a relative file and the record number is not known. It is possible to read each record and compare each for the desired name. But this is very time consuming if the file has many records.

If the file is kept in name order, the records can be searched using an alternative method. This method is called a binary search. In order to use a binary search, the relative file must be arranged in sorted order. Using the above example, relative record 1 must contain a name with the lowest collating sequence while the last relative record must contain a name with the highest collating sequence. Thus the name AARON might be contained in relative record 1 and ZYPHER might be contained in the last relative record of

## Anatomy of the 1541 Disk Drive

the file and all other names would be ordered throughout.

When records are added to the file, then the records must be reordered. Similarly if a name is changed, then the records must be reordered.

The binary search can be explained using a simple example. When you want to find a name in the telephone book, you don't search through it sequentially. You open the book in the middle and compare the first letter of the desired name with the first letter of names on the page. If the desired name comes before these, you turn halfway into the first section of the book, and so on. You go through it systematically.

The binary search is not a sequential search. It identifies a record halfway through the remaining number of records. The following example will clarify this:

There exists the following relative file, sorted in ascending order:

Record number	Contents
1	1985
2	1999
3	2005
4	2230
5	2465
6	2897
7	3490
8	3539
9	4123
10	5000
11	5210
12	6450
13	6500
14	6550
15	6999

Out of these fifteen records we will search for a contents of 3490. It is not known which record it is stored in.

We must first know how many records are in the file. In this case, there are fifteen. We divide this by two. The middle of the file is record eight with the contents 3539. We determine if the contents of this record equal to the target value, and if not, whether it is larger or smaller. In this case, it (3539) is larger. This means the record we are looking for is in the first half of the file. So we divide eight by two and examine the contents of record four, 2230. Since 2230 is less than 3490, it lies between four and eight. We again divide by two and add this to record 4 which and results in record 6 whose contents is 2897. 2897 is less than 3490, so our target lies between records six and eight. Record seven is indeed the record we are looking for.

The principle of the binary search is to determine by the result of each comparison whether to search upwards or downwards until the search data is found. The maximum number of comparisons can be found using the following formula:

$$S = \text{INT}(\text{LOG}(N) / \text{LOG}(2) + 1)$$

S is the number of comparisons (searches) and N is the number of records in the file. In a sorted relative data file with 1000 records, no more than ten comparisons will be necessary to find the desired record!

Let's create a relative data file with fifteen records to test the binary search:

```
100 OPEN1,8,2,"BINARY.REL,L,"+CHR$(5)
110 FORI=1TO15
120 READ RC$
130 PRINT#1,RC$
140 NEXT I
150 CLOSE 1:CLOSE 2:END
160 DATA 1985,1999,2005,2230,2465,2897,3490,3539
170 DATA 4123,5000,5210,6450,6500,6550,6999
```

This program puts the fifteen records in a file called BINARY.REL using the values given in lines 160-170. The position command is not necessary because the data will be written straight through from first to last record. After opening the file the pointer points to the first record. This file is designed to be searched with the binary method. The following program is based on the logic of the binary search:

```
100 OPEN1,8,2,"BINARY.REL,L,"+CHR$(5)
110 OPEN2,8,15
120 PRINTCHR$(147)
140 N=15: REM NUMBER OF RECORDS
150 I=LOG(N)/LOG(2)
160 IF I-INT(I)>0 THEN I=INT(I)+1
170 M=I-1
180 I=2^I
190 X=I/2
210 INPUT"RECORD TO FIND (* TO END): ";SR$
220 IF SR$="" THEN 320
230 IF M<0 THEN PRINT"RECORD NOT FOUND":GOTO140
240 M=M-1
250 PRINT#2,"P"+CHR$(2)+CHR$(X)+CHR$(0)+CHR$(1)
260 INPUT#1,RC$
270 IF SR$=RC$ THEN 340
280 IF SR$<RC$ THEN X=X-2^M:GOTO230
290 X=X+2^M
300 IF X>I THEN PRINT"END OF FILE EXCEEDED!"
310 GOTO 230
320 CLOSE 1:CLOSE 2
```



## Anatomy of the 1541 Disk Drive

```
330 END
340 PRINT"RECORD FOUND!"
350 PRINT"CONTENTS : ";RCS
360 GOTO 140
```

### Program Documentation:

```
100      The relative file "BINARY.REL" is opened.
110      The command channel is opened.
120      The screen is erased.
140      The number of records is assigned to the variable
        N.
150-190  If the maximum number of records does not
        represent a power of two, the next higher power
        of two is formed. The file will be expanded, but
        no records are lost. The exponent of this power of
        two is used as the index. X is the value of I/2.
        I/2 indicates the exact middle of the (expanded)
        file. After that, the variable M receives the
        value of I-1.
210-220  The record to be found is read. To end the
        program, enter a '*'.
230      If M<0, the record was not found.
240      M is decremented by one. The next Mth power
        represents half of the rest of the file.
250-260  The file is positioned over the record containing
        in the variable X.
270      If the target record is found, the search is
        ended and the record displayed.
280-310  It is determined if the target record is larger
        or smaller than the record just read. The middle
        of the upper or lower half (as appropriate) is
        stored in the variable X.
320-330  The file is closed and the program is ended.
340-360  The found record is displayed.
```

This binary search, coded in BASIC, is implemented universally. Only the number of records and the appropriate record to be searched for need be changed. You can use this routine for finding records in your sorted relative data files.

### 1.5.8 Searching Records with a Separate Index File

If you work with individual records frequently and need quick access with alphanumeric keys that don't correspond to the logical record number, and your file is not sorted, we recommend another method.

Create an index file for each desired key field, in which each record is composed of

- an index key
- the corresponding record number

This entire index file is to be loaded into the computer's memory. An example:

You have constructed your name and address manager as a relative file consisting of

- First name
- Last name
- Street
- City, State
- Zip code
- Telephone number

You want to be able to search the file based on the last name. So you create an additional sequential file that contains the desired key (in this case the last name) and the record number of the corresponding record in the relative file.

The index file is read completely into the computer so the search can be accomplished as quickly as possible. If you want to access a record that has the last name **HARRIS**, then you search through the appropriate index in memory and when found, read the corresponding relative record by using the record number also contained in the index.

Here is an example:

We assume that a data file and an index file exist for the names:

Data file:			Index file:		
=====			=====		
Last name	First name	more fields	Index (last name)	Record No.	
				LB	HB
Smith	John	.....	Smith	01	00
Harris	Sam	.....	Harris	02	00
Hanson	Carl	.....	Hanson	03	00
Johnson	Mark	.....	Johnson	04	00
.	.		.	.	
.	.		.	.	
.	.		.	.	
Green	Simon	.....	Green	99	00

The file contains 99 records. Before the program can be used, the index file must be read in. This can be a sequential file, which can be read into a memory table reserved with DIM ITS(99). The first twenty characters of each index table position comprise the last name. The next

## Anatomy of the 1541 Disk Drive

to the last byte (no. 21) is the low byte and the last byte (no. 22) is the high byte of the record number. With these conditions, a desired record can be found with the following routine:

```
.
.
100 INPUT "LAST NAME";N$
110 FOR I=1 TO 99
120 IF LEFT$(ITS(I),20)=N$ THEN 150
130 NEXT I
140 PRINT "NAME NOT FOUND!":END
150 PRINT "RECORD FOUND!"
160 OPEN1,8,2,"ADDRESS,L,"+CHR$(81)
170 OPEN 2,8,15
180 PRINT#2,"P"+CHR$(2)+MID$(ITS(I),21,1)+CHR$(0)
    +CHR$(1)
190 INPUT#1,RCS
.
.
```

The loop in lines 110-130 goes through the index table sequentially, searching for the target name contained in the twenty leftmost characters. If the name is not found, an appropriate message is given (line 140), before the program is ended.

If, in line 120, the target name matches the index entry, the program branches to line 150. After giving the message, the address file is opened. After opening the command channel, the position command is sent to the disk. Because the next to the last byte of the index entry contains the low byte of the record number, it must be extracted using the MID\$ function. The high byte is known to be zero since there are fewer than 255 record.

Finally the relative record is read in line 190.

The access of index files is an equally fast and extraordinarily flexible form of data organization. One can theoretically have as many index files as desired. Above all, you must take note of two important restrictions:

1. Changes in the main data file which affect the key fields must also be made to the corresponding index file. With several index files this can become very time-consuming.
2. The number and size of the index files that are kept in the computer's memory for fast access are limited by the availability of memory.

### 1.5.9 Changing Records

The logical process for changing a record is this:

1. Read the record
2. Split the record into its fields
3. Change the appropriate field
4. Rebuild the record (combine fields)
5. Rewrite the record

In section 1.5.5 we wrote some records in the file "TEST.REL". This file had the following properties:

```
Record length           41 bytes
Number of records       100
Number of fields         4
Length, position field 1 : 10, 1-10
" , " field 2 : 5, 11-15
" , " field 3 : 10, 16-25
" , " field 4 : 15, 26-40
Trailing RETURN in position 41
```

A file description such as the one above should be made for each of your files. This is very important if other programs are to use these data. The file description defines the order and length of the fields of the file.

In this file, we allow for the contents of the records to be changed. The following program allows changes:

```
100 REM =====
110 REM     PREPARATION
120 REM =====
130 BL$=" "
140 OPEN 1,8,2,"TEST.REL,L,"+CHR$(41)
150 OPEN 2,8,15
160 REM =====
170 REM     READ RECORD
180 REM =====
190 PRINT CHR$(147)
200 INPUT"RECORD NUMBER (1-100): ";RN
205 IF RN<1 OR RN>100 THEN PRINTCHR$(145);:GOTO200
210 PRINT"-----"
220 PRINT#2,"P"+CHR$(2)+CHR$(RN)+CHR$(0)+CHR$(1)
230 INPUT#1,RCS
240 IF ASC(RCS)<>255 THEN 270
250 PRINT "RECORD NOT WRITTEN"
260 GOTO 630
270 REM =====
280 REM     PREPARE RECORD
290 REM =====
300 F$(1)=MID$(RCS,1,10)
310 F$(2)=MID$(RCS,11,5)
320 F$(3)=MID$(RCS,16,10)
330 F$(4)=MID$(RCS,26,15)
```

## Anatomy of the 1541 Disk Drive

```
340 REM =====
350 REM      DISPLAY FIELDS
360 REM =====
370 PRINT CHR$(147)
380 FOR I=1 TO 4
390 PRINT"FIELD";I;": ";F$(I)
400 NEXT I
410 PRINT"-----"
420 REM =====
430 REM      CHANGE FIELDS
440 REM =====
450 PRINT"CHANGE WHICH FIELD (1-4)?"
460 GETX$:IF X$<"1" OR X$>"4" THEN 460
470 INPUT"NEW CONTENTS : ";F$(VAL(X$))
480 PRINT"RECORD IS CHANGED"
490 PRINT"MORE CHANGES IN THIS RECORD (Y/N)?"
500 GETX$:IF X$<>"Y" AND X$<>"N" THEN 500
510 IF X$="Y" THEN 340
520 REM =====
530 REM      CHAIN FIELDS
540 REM =====
550 RC$=F$(1)+LEFT$(BL$,10-LEN(F$(1)))
560 RC$=RC$+F$(2)+LEFT$(BL$,5-LEN(F$(2)))
570 RC$=RC$+F$(3)+LEFT$(BL$,10-LEN(F$(3)))
580 RC$=RC$+F$(4)+LEFT$(BL$,15-LEN(F$(4)))
590 REM =====
600 REM      WRITE RECORD BACK
610 REM =====
620 PRINT#1,RC$
630 REM =====
640 REM      END PROGRAM?
650 REM =====
660 PRINT"MORE CHANGES TO FILE (Y/N)?"
670 GETX$:IF X$<>"Y" AND X$<>"N" THEN 670
680 IF X$="Y" THEN 160
690 CLOSE 1:CLOSE 2:END
```

After this program is RUN you can change any desired record. This record must have been written with the program in section 1.5.5.

This editing program does not check the new field data for correct length.

The important commands in this program have already been explained in the corresponding sections.

### 1.5.10 Expanding a Relative File

Every relative file has a user-determined number of records that ranges from 1 to 65538. This number is the record with the highest record number and is written to the file with a

value of CHR\$(255). Writing this last record also formats all records in the file that precede this record number with CHR\$(255).

You can expand the size of a relative file at a later time. For example, consider a relative file that is initially created with three records. After the file is OPENed, you position the file at record number 3 and write the record with CHR\$(255). Here's an example of how you might do this:

```
10 OPEN 1,8,2,"RELFILE,L,"+CHR$(50)
20 OPEN 15,8,15
30 PRINT#15,"P"+CHR$(2)+CHR$(3)+CHR$(0)+CHR$(1)
40 PRINT#1,CHR$(255)
```

When statement 40 is performed, not only is record 3 written, but records 1 and 2 are also formatted by the DOS. Subsequently, if you position and write a 90th record, the DOS formats records 4 through 89 (see lines 150 and 160 below). Each time the file is expanded, the DOS formats records between the current high record number and the new high record number.

```
150 PRINT#15,"P"+ CHR$(2)+CHR$(90)+CHR$(0)+CHR$(1)
160 PRINT#1,CHR$(255)
.
.
500 PRINT#15,"P"+CHR$(2)+CHR$(175)+CHR$(0)+CHR$(1)
510 PRINT#1,CHR$(255)
.
.
```

An existing relative file can be expanded at any time, provided there is sufficient room on the disk. To do so, the new last record is written with CHR\$(255). At the same time, all records between the old and new end of file are also formatted.

When writing a record to a relative file whose record number is higher than the current high record number, a DOS error is not returned. If there is room on the diskette for the new records (current high record number through the new high record number) the file is simply expanded. If there is a lack of space on the diskette for the new records, the DOS error **FILE TOO LARGE** is returned. When reading a record from a relative file whose record number is higher than the current high record number, the DOS error **RECORD NOT PRESENT** is returned to the error channel.

### 1.5.11 Home Accounting with Relative Data Storage

A complete example of problem solving using relative files offers you a good insight into the organization of relative file processing. It can be used by most readers of this book. Few examples of relative file usage have been explained elsewhere, so here is such a program.

In this application, individual accounts are numbered. This account number is used as a key to the corresponding records.

This provides that each account contain a clear text description. The first field of each record is this account name. Twenty characters are allowed for the name.

Since information is needed for each month, twelve fields are necessary for each record. These summary fields are each ten characters long. The account summaries are stored as strings which are converted to numbers with the help of the VAL function. The record consists of 141 characters (twenty for the name, 12\*10 for the month summaries and one for RETURN).

The layout of the records follows:

Field	Length	Position
Account name	20	1-20
January summary	10	21-30
February summary	10	31-40
.		
.		
.		
.		
November summary	10	121-130
December summary	10	131-140

The maximum number of accounts per year is set to twenty. Therefore, a year's file consists of twenty records of 141 bytes each.

We also specified the functions that this program is to perform.

- \* Create accounts
- \* Post to accounts
- \* Display summary by Account
- \* Display account names
- \* Display Monthly summary

\* Display Year-end summary

Create accounts:

-----  
This function creates the file for a year. It asks for the number and names of the accounts. The records are then written with the account name and the summary fields are set to zero. Should a data file already exist with the same name, the old file is deleted.

Post to accounts:

-----  
This function asks for the account number to be posted and whether the posting is an income or expense. For example, the category "SALARY" is an income account and the category "RENT" is an expense account.

After this, the current contents of the account are displayed. When you post the appropriate amount, which is always positive. If you are making a correction entry, use a negative amount.

Now the updated contents are displayed. You may then make a new entry.

Producing account summary:

-----  
After entering the account number, the summary of the twelve months and the year's total are displayed for that account.

Display account names:

-----  
Each account is determined by its number. Should you forget a number, this function lists all accounts by name and corresponding number.

Display monthly summary:

-----  
Here the income or expenses of all accounts are displayed. The monthly balance of all accounts is also displayed.

Display year-end summary:

-----  
This function shows the summary of all accounts and the year-end balance. This display takes some time, since all monthly fields of each record must be read and totaled. It accesses the entire file.

Here's the program listing:



# Anatomy of the 1541 Disk Drive

```

100 POKE 53280,2:POKE53281,2:PRINTCHR$(158);:
    BLS="":DIMS(12)
110 GOSUB 2050
120 INPUT"CURRENT YEAR : ";Y$
130 IF Y$<"1984"ORY$>"1999"THENPRINTCHR$(145);:GOTO120
140 GOSUB 2050
150 PRINT"SELECT A FUNCTION:
160 PRINT"-----":PRINT
170 PRINT"    -1- CREATE ACCOUNTS"
180 PRINT"    -2- POST TO ACCOUNTS"
190 PRINT"    -3- ACCOUNT SUMMARY"
200 PRINT"    -4- DISPLAY ACCOUNT NAMES"
210 PRINT"    -5- MONTHLY SUMMARY"
220 PRINT"    -6- YEAR SUMMARY":PRINT
230 PRINT"    -0- END PROGRAM"
240 GETX$:IFX$<"0"ORX$>"9"THEN240
250 IFX$<>"0"THEN270
260 END
270 ONVAL(X$)GOSUB 290,560,920,1160,1370,1720
280 GOTO 140
290 REM =====
300 REM    CREATE ACCOUNTS
310 REM =====
320 GOSUB 2050
330 PRINT"CAUTION! ANY PREVIOUS FILE FOR THIS YEAR"
340 PRINT"WILL BE ERASED!":PRINT
350 PRINT"CONTINUE (Y/N)?"
360 GETX$:IFX$<>"Y"ANDX$<>"N"THEN360
370 IFX$="Y"THEN390
380 CLOSE1:CLOSE2:RETURN
390 OPEN2,8,15,"S:ACCOUNTS"+Y$
400 OPEN1,8,2,"ACCOUNTS"+Y$+",L,"+CHR$(141)
410 GOSUB 2050
420 INPUT"HOW MANY ACCOUNTS (1-20): ";AN
430 PRINT
440 IFAN<1ORAN>20THENPRINTCHR$(145);:GOTO420
450 FORI=1TOAN
460 PRINT"NAME OF ACCOUNT NO.";I;": ";
470 INPUTAN$
480 IFLEN(AN$)>20THENPRINTCHR$(145);:GOTO420
490 RCS=AN$+LEFT$(BLS,20-LEN(AN$))
500 FORX=1TO12
510 RCS=RCS+STR$(0)+LEFT$(BLS,8)
520 NEXTX
530 PRINT#1,RCS
540 NEXT I
550 CLOSE 1:CLOSE 2:RETURN
560 REM =====
570 REM    POSTING
580 REM =====
590 GOSUB2050
600 INPUT"ACCOUNT NUMBER";AN
610 IFAN<1ORAN>20THENPRINTCHR$(145);:GOTO600
620 GOSUB2140
630 PRINT"-----"

```

```

640 PRINT"NO.";AN;" - ";AN$
650 PRINT"-----"
660 PRINT"INCOME OR EXPENSE (I/E)?"
670 PRINT"-----"
680 GETX$:IFX$<>"I"ANDX$<>"E"THEN680
690 INPUT"MONTH (1-12)      : ";M
700 IFM<1ORM>12THENPRINTCHR$(145);:GOTO690
710 PRINT"-----"
720 PRINT"OLD CONTENTS      : ";S(M)
730 PRINT"-----"
740 INPUT"POSTING AMOUNT : ";PA
750 PRINT"-----"
760 IFX$="I"THENS(M)=S(M)+PA:GOTO780
770 S(M)=S(M)-PA
780 PRINT"NEW CONTENTS      : ";S(M)
790 PRINT"-----"
800 RC$=AN$+LEFT$(BL$,20-LEN(AN$))
810 FORI=1TO12
820 S$=STR$(S(I))
830 RC$=RC$+S$+LEFT$(BL$,10-LEN(S$))
840 NEXTI
850 PRINT#2,"P"+CHR$(2)+CHR$(AN)+CHR$(0)+CHR$(1)
860 PRINT#1,RC$
870 CLOSE1:CLOSE2
880 PRINT"FURTHER POSTING (Y/N)?"
890 GETX$:IFX$<>"Y"ANDX$<>"N"THEN890
900 IFX$<>"Y"THENGOSUB2050:GOTO600
910 RETURN
920 REM =====
930 REM   ACCOUNT SUMMARY
940 REM =====
950 GOSUB2050
960 INPUT"ACCOUNT NUMBER : ";AN
970 IFAN<1ORAN>20THENPRINTCHR$(145);:GOTO960
980 GOSUB2140
990 GOSUB2050:PRINTCHR$(145);CHR$(145);
1000 PRINT"-----"
1010 PRINT"NO.";AN;" - ";AN$
1020 PRINT"-----"
1030 PRINT"MONTH TOTAL"
1040 PRINT"-----"
1050 TL=0
1060 FORI=1TO12
1070 PRINTI;TAB(8);S(I)
1080 TL=TL+S(I)
1090 NEXTI
1100 PRINT"-----"
1110 PRINT"TOTAL";TAB(8);TL
1120 PRINTTAB(9);"====="
1130 PRINT"RETURN FOR MORE"
1140 INPUTX$
1150 CLOSE1:CLOSE2:RETURN
1160 REM =====
1170 REM DISPLAY ACCOUNT NAMES
1180 REM =====

```

# Anatomy of the 1541 Disk Drive

```

1190 GOSUB2050
1200 OPEN1,8,2,"ACCOUNTS"+Y$+"",L,"+CHR$(141)
1210 OPEN2,8,15
1220 I=1
1230 PRINT#2,"P"+CHR$(2)+CHR$(I)+CHR$(0)+CHR$(1)
1240 RC$=""
1250 FORX=1TO20
1260 GET#1,X$
1270 RC$=RC$+X$
1280 NEXTX
1290 INPUT#2,X
1300 IFX=50THEN1340
1320 PRINTI;" - ";RC$
1330 I=I+1:GOTO1230
1340 PRINT"RETURN FOR MORE"
1350 INPUTXS
1360 CLOSE1:CLOSE2:RETURN
1370 REM =====
1380 REM MONTH SUMMARY
1390 REM =====
1400 GOSUB2050
1410 INPUT"MONTH : ";M
1420 GOSUB2050
1430 PRINT"-----"
1440 PRINT"NO. NAME CONTENTS"
1450 PRINT"-----"
1460 OPEN1,8,2,"ACCOUNTS"+Y$+"",L,"+CHR$(141)
1470 OPEN2,8,15
1480 TL=0
1490 FORAN=1TO20
1500 AN$="":S$=""
1510 PRINT#2,"P"+CHR$(2)+CHR$(AN)+CHR$(0)+CHR$(1)
1520 FORI=1TO20
1530 GET#1,X$
1540 AN$=AN$+X$
1550 NEXTI
1560 INPUT#2,F
1570 IFF<>50THEN1590
1580 GOTO1670
1590 PRINT#2,"P"+CHR$(2)+CHR$(AN)+CHR$(0)+CHR$(20+(M-1)*10)
1600 FORI=1TO10
1610 GET#1,X$
1620 S$=S$+X$
1630 NEXT I
1640 TL=TL+VAL(S$)
1650 PRINT AN;TAB(6);AN$;TAB(26);S$
1660 NEXT AN
1670 PRINT"-----"
1680 PRINT"TOTAL BALANCE";TAB(26);STR$(TL)
1690 PRINTTAB(26);"=====
1700 PRINT"RETURN FOR MORE";
1710 INPUTXS:CLOSE1:CLOSE2:RETURN
1720 REM =====
1730 REM YEAR SUMMARY
1740 REM =====

```

```

1750 GOSUB2050
1760 OPEN1,8,2,"ACCOUNTS"+Y$+",L,"+CHR$(141)
1770 OPEN2,8,15
1780 PRINT"-----"
1790 PRINT"NO. NAME YEAR BALANCE"
1800 PRINT"-----"
1810 TL=0
1820 FOR AN=1TO20
1830 PRINT#2,"P"+CHR$(2)+CHR$(AN)+CHR$(0)+CHR$(1)
1840 RC$=""
1850 FORI=1TO140
1860 GET#1,X$
1870 RC$=RC$+X$
1880 NEXTI
1890 INPUT#2,F:IFF=50THEN1980
1900 AN$=LEFT$(RC$,20)
1910 YB=0
1920 FORI=1TO10
1930 YB=YB+VAL(MID$(RC$,20+(I-1)*10,10))
1940 NEXTI
1950 TL=TL+YB
1960 PRINTAN;TAB(6);AN$;TAB(26);YB
1970 NEXTAN
1980 PRINT"-----"
1990 CLOSE1:CLOSE2
2000 PRINT"TOTAL BALANCE";TAB(26);TL
2010 PRINTTAB(26);"====="
2020 PRINT"RETURN FOR MORE"
2030 INPUTX$
2040 RETURN
2050 REM =====
2060 REM PROGRAM HEADING
2070 REM =====
2080 PRINTCHR$(147);
2090 PRINTTAB(4);"=====
2100 PRINTTAB(4);"H O M E A C C O U N T I N G"
2110 PRINTTAB(4);"=====
2120 PRINT:PRINT
2130 RETURN
2140 REM =====
2150 REM READ ACCOUNT
2160 REM =====
2170 OPEN1,8,2,"ACCOUNTS"+Y$+",L,"+CHR$(141)
2180 OPEN2,8,15
2190 PRINT#2,"P"+CHR$(2)+CHR$(AN)+CHR$(0)+CHR$(1)
2200 RC$=""
2210 FORI=1TO140
2220 GET#1,X$
2230 RC$=RC$+X$
2240 NEXT I
2250 INPUT#2,F
2260 IFF<>50THEN2300
2270 PRINT"YEAR FILE OR ACCOUNT NOT FOUND!":PRINT
2280 PRINT"RETURN FOR MORE":INPUTX$
2290 CLOSE1:CLOSE2:RETURN

```

## Anatomy of the 1541 Disk Drive

```
2300 AN$=LEFT$(RC$,20)
2310 TL=0
2320 FORI=1TO12
2330 S(I)=VAL(MID$(RC$,20+(I-1)*10,10))
2340 TL=TL+S(I)
2350 NEXT I
2360 RETURN
```

### Program Documentation:

#### Initialization:

```
-----
100      Screen and character color set; blank character
        string defined; variable for account summaries
        dimensioned.
110-130  Program heading displayed and current year read.
140-280  Program functions displayed and choice read;
        corresponding subprogram called.
```

#### Establish Accounts:

```
-----
390-400  Any existing files of this year are erased and the
        new file is opened.
480      Account name is placed in positions 1-20 of the
        record RC$.
500-540  Month summaries are set to zero and placed in the
        record as string variables.
530      The record is transferred with a trailing RETURN.
```

#### Posting:

```
-----
590      The routine "Read Account" is called. This routine
        places the month summaries of the account in the
        variables S(1) to S(12).
800      Account name is placed in the record.
810-840  Account summary is placed in the record.
850-860  Record is transferred.
```

#### Account Summary:

```
-----
980      Desired account is read and the month summaries
        are placed in variables S(1) to S(12).
1050-1090 Month summaries are displayed and the total (TL)
        is added up.
1110     Total displayed.
```

#### Display Account Names:

```
-----
1220     Account number is initialized.
1230     The head is positioned over the corresponding
```

record.

1240-1280 Account name is read out of the record in RC\$.  
 1290-1300 If RECORD NOT PRESENT is sent over the error  
 channel (error 50), the routine is broken off.  
 1320 Account number and name are displayed.

## Month Summary:

-----  
 1490-1660 Loop to read all accounts.  
 1510 Position head over record.  
 1520-1550 Read account name.  
 1560-1580 Determine if account exists; stop if all twenty  
 accounts have been defined.  
 1590 Position over summary field of the desired month.  
 1600-1630 Read the month summary.  
 1640 Add month summary to total.  
 1650 Account number, account name and month summary are  
 displayed.  
 1680 Total balance displayed.

## Year Summary:

-----  
 1820-1970 Loop to read all accounts  
 1830 Position head over record.  
 1850-1880 Complete record read into RC\$.  
 1890 Test if RECORD NOT PRESENT.  
 1900 Get account name from record.  
 1920-1940 Read month summary, convert to numerical form and  
 add to year summary (YS).  
 1950 Year summary (YS) is added to total (TL).  
 1960 Account number, account name and year summary  
 displayed.  
 2000 Total balance (month balance) displayed.

## Read Account:

-----  
 2190 Position over record given in AN.  
 2210-2240 Read record into RC\$.  
 2250-2260 Test if RECORD NOT PRESENT.  
 2300 Account name read from record.  
 2320-2350 Month summaries read from record, converted to  
 numerical form and placed into the table S(1) to  
 S(12).

## Anatomy of the 1541 Disk Drive

### 1.6 Disk Error Messages and their Causes

If you cause an error while working with the disk drive, the drive signals this by blinking the red LED. The LED blinks until you read the error channel of the disk drive or until you send a new command. First we want to see how to read the error message from the disk drive.

In order to do this, the error/command channel must be opened with the secondary address 15:

```
100 OPEN 15,8,15
110 INPUT#15,A,B$,C,D
120 PRINT A,B$,C,D
```

If no error has occurred, the following is displayed:

```
0      OK      0      0
```

The first number is the error number, in this case zero, which means no error has occurred. Next follows the error message (variable B\$). The variables C and D contain the track and sector numbers, respectively, in which the error occurred, which is dependent on the type of error (mainly associated with hardware errors and block-oriented commands).

This routine accomplishes the same function:

```
100 OPEN15,8,15
110 GET#15,AS:PRINTA$;:IFST<>64THEN110

00, OK,00,00
```

Here characters are read from the error channel until the end is recognized (status = 64). This gives the error message exactly as the BASIC 4.0 command

#### PRINT DS\$

When using BASIC 4.0, variables DS\$ and DS are reserved variables which contain the complete error message and error number. Each access of these variables gives the error status of the last disk operation. Unfortunately, the Commodore 64 does not use BASIC 4.0, so these variables are meaningless in Commodore 64 BASIC (BASIC 2.0).

Next follows the list of error messages that the DOS can recognize:

```
00, OK,00,00
```

This message occurs when the last disk operation was error free or if no command or data was sent after the last error message.

## 01,FILES SCRATCHED,XX,00

This is the message after a SCRATCH command. The number XX denotes the number of files that were erased. Since this is not really an error message, the LED does not blink.

## 20,READ ERROR,TT,SS

This error means that the 'header' of a block was not found. It is usually the result of a defective diskette. TT and SS designate the track and sector in which the error occurred. Remedy: change defective diskette.

## 21,READ ERROR,TT,SS

This is also a read error. The SYNC (synchronous) marker of a block was not found. The cause may be an unformatted disk, or no disk in the drive. This error can also be caused by a misaligned read/write head. Remedy: Either insert a diskette, format the disk, or have the read/write head aligned.

## 22,READ ERROR,TT,SS

This error message means that a checksum error has occurred in the header of a data block, which can be caused by the incorrect writing of a block.

## 23,READ ERROR,TT,SS

The error implies that a data block was read into the DOS buffer, but a checksum error occurred. One or more data bytes are incorrect. Remedy: Save as many files as possible onto another diskette.

## 24,READ ERROR,TT,SS

This error also results from a checksum error in the data block or in the preceding data header. Incorrect bytes have been read. Remedy: same as error 23.

## 25,WRITE ERROR,TT,SS

This error is actually a VERIFY ERROR. After writing every block the data is read again checked against the data in the buffer. This error is produced if the data are not identical. Remedy: Repeat the command that caused the error. If this doesn't work, the corresponding block must be locked out from further use with the block-allocate command.

## 26,WRITE PROTECT ON,TT,SS

An attempt was made to write to a disk with a write protect tab on it. Remedy: Remove write protect tab.

## 27,READ ERROR,TT,SS

A checksum error occurred in the header of a data block. Remedy: Repeat command or rescue block.



## Anatomy of the 1541 Disk Drive

### 28,WRITE ERROR,TT,SS

After writing a data block, the SYNC characters of the next data block were not found. Remedy: Format disk again, or exchange it.

### 29,DISK ID MISMATCH,TT,SS

The ID (two character disk identification) in the DOS memory does not agree with the ID on the diskette. The diskette was either not initialized or there is an error in the header of a data block. Remedy: Initialize diskette.

### 30,SYNTAX ERROR,00,00

A command was sent over the command channel that the DOS could not understand. Remedy: Check and correct command.

### 31,SYNTAX ERROR,00,00

A command was not recognized by the DOS, for example, the BACKUP command (Duplicate) on the 1541. Remedy: Do not use the command.

### 32,SYNTAX ERROR,00,00

The command sent over the command channel was longer than 40 characters. Remedy: Shorten command.

### 33,SYNTAX ERROR,00,00

A wildcard ('\*' or '?') was used in an OPEN or SAVE command. Remedy: Remove wildcard.

### 34,SYNTAX ERROR,00,00

The DOS cannot find the filename in a command. This may be because a colon was forgotten after the command word. Remedy: Check and correct command.

### 39,FILE NOT FOUND,00,00

User program of type 'USR' was not found for automatic execution. Remedy: Check filename.

### 50,RECORD NOT PRESENT,00,00

A record was addressed in a relative data file that has not yet been written. When writing a record this is not really an error. You can avoid this error message if you write the highest record number of the file with CHR\$(255) when initializing it. This error will no longer occur upon later access.

### 51,OVERFLOW IN RECORD,00,00

The number of characters sent when writing a record in a relative file was greater than the record length. The excess characters are ignored.

### 52,FILE TOO LARGE,00,00

The record number of a relative file is too big; the diskette does not have enough capacity. Remedy: Use another diskette or reduce the record number.

**60,WRITE FILE OPEN,00,00**

An attempt was made to OPEN a file that had not previously been CLOSED after writing. Remedy: Use mode 'M' in the OPEN command to read the file.

**61,FILE NOT OPEN,00,00**

A file was accessed that had not been OPENed. Remedy: Open the file or check the filename.

**62,FILE NOT FOUND,00,00**

An attempt was made to load a program or open a file that does not exist on the diskette. Remedy: Check the filename.

**63,FILE EXISTS,00,00**

An attempt was made to establish a new file with the name of a file already on the diskette. Remedy: Use a different filename or @: (to replace the old file).

**64,FILE TYPE MISMATCH,00,00**

The file type use in the OPEN command does not agree with the file type in the directory. Remedy: Correct file type.

**65,NO BLOCK,TT,SS**

This error message is given in association with the BLOCK-ALLOCATE command when the specified block is no longer free. In this case, the DOS automatically searches for a free block with a higher sector and/or track number and gives these values as the track and sector number in the error message. If no block with a greater number is free, two zeroes will be given.

**66,ILLEGAL TRACK OR SECTOR,TT,SS**

If you attempt to use a block with the block commands that does not exist, this error is returned.

**67,ILLEGAL TRACK OR SECTOR,TT,SS**

The track-sector combination of a file produces a non-existent track or sector.

**70,NO CHANNEL,00,00**

An attempt was made to open more files than channels available or a direct access channel is already reserved.

**71,DIR ERROR,TT,SS**

The number of free blocks in the DOS storage does not agree with the BAM. Usually this means the disk has not been initialized.

**72,DISK FULL,00,00**

Fewer than three blocks are free on the diskette or the maximum number of directory entries have been used (144 on the VIC 1541).

## **Anatomy of the 1541 Disk Drive**

### **73,CBM DOS V.26 1541,00,00**

The message is the power-up message of the VIC 1541. As an error message, it appears when an attempt is made to write to a disk that was not formatted with the same DOS version, for example, the forerunner of the CBM 4040, the CBM 2040 (DOS version 1.0).

### **74,DRIVE NOT READY,00,00**

When one attempts to use the disk without a diskette in the drive, this error message is returned.

### **75,FORMAT SPEED ERROR,00,00**

This error message occurs only on the CBM 8250. It indicates a deviation from the normal revolutions per minute while formatting.

### 1.7 Overview of Commands with a Comparison of BASIC 2.0 - BASIC 4.0 - DOS 5.1

BASIC 2.0	BASIC 4.0 (abbrev)	DOS 5.1
OPEN - Mode 'A'	APPEND (aP)	
	BACKUP (bA)	
LOAD"\$",8 & LIST	CATALOG (cA)	@\$ or >\$
V(alidate)	COLLECT (coL)	@V or >V
	CONCAT (conC)	
C(opy)	COPY (coP)	@C:... or >C:..
CLOSE ...	DCLOSE (dC)	
LOAD"...",8	DLOAD (dL)	@file or /file
OPEN ...,8,...	DOPEN (dO)	
OPEN 1,8,15 ...	D\$\$, DS	@ or >
SAVE"...",8	DSAVE (dS)	
N(ew)	HEADER (hE)	@N:... or >N:..
I(nitalize)	I(initialize)	@I or >I
P	RECORD (reC)	
R(ename)	RENAME (reN)	@R:... or >R:..
S(cratch)	SCRATCH (sC)	@S:... or >S:..

This table lists the different versions of BASIC. The DOS 5.1 is found on the TEST/DEMO disk and will be described in section 4.2.1.

The essential difference between BASIC 2.0 and BASIC 4.0 is that with BASIC 2.0, each command is executed by the disk control system (DOS) and must be sent over channel 15. The disk commands of BASIC 4.0 manage this channel themselves (with the exception of INITIALIZE). For example, the command HEADER D0,"DISK1",IHJ generates the same sequence of commands necessary in BASIC 2.0, namely:

```
OPEN 1,8,15,"N:DISK1,HJ"
CLOSE 1
```

Here are the specifics of the BASIC 4.0 commands:

Note the following parameters:

```
lfn = logical file number
dn  = drive number - drive 0 (D0) or drive 1 (D1) with
      a double drive, or D0 for a single drive
da  = device address of the disk drive (U4 to U31)
```

Information in parentheses is optional. The standard parameters D0 and U8 will be used (meaning Drive 0 and Unit 8).

## Anatomy of the 1541 Disk Drive

### APPEND:

This command allows data to be added to a sequential file, which is accomplished in BASIC 2.0 with the OPEN-command mode A.

This command has the following format:

```
APPEND#lfn,"filename"(:,Ddn,Uda)
```

For example, should the sequential file "SEQU.1" be on drive 0, the following statements are necessary to add a data record to it:

```
100 APPEND#1,"SEQU.1",D0
110 PRINT#1,X$
120 CLOSE 1
```

### BACKUP:

With this command, a complete diskette can be copied. The BACKUP command can only be used with a dual disk drive (such as the 4040), however. Notice the format of this command:

```
BACKUP Ddn TO Ddn(:,Uda)
```

It is important that either D0 to D1 or D1 to D0 be given. An example:

The diskette in drive 1 is supposed to be copied onto the disk in drive 0. To this end, give the following command:

```
BACKUP D1 TO D0
```

### CATALOG:

The CATALOG command of BASIC 4.0 has the advantage that the program in the computer's memory is not erased, as is true in BASIC 2.0. The format of the command:

```
CATALOG (Ddn,Uda)
```

If no drive number is given for a double drive, the contents of both drives are given. With a single drive, CATALOG D0 is assumed. An example:

```
CATALOG D0
```

The contents of the disk in drive 0 will be displayed.

### COLLECT:

This command corresponds with the VALIDATE command of BASIC 2.0. The syntax of this command looks like this:

```
COLLECT (Ddn)
```

## CONCAT:

**CONCAT** concatenates sequential files, in which one file is to be made from the data of two files. The format:

```
CONCAT (Ddn,)"file1" TO (Ddn,)"file2" (ON Uda)
```

Suppose you want to combine the data of the files "SEQU.2" in drive 0 and "SEQU.1" in D1. To accomplish this, issue the following command:

```
CONCAT D0,"SEQU.2" TO D1,"SEQU.1"
```

## COPY:

With this command files can be copied from one drive to the other (except relative files). The command is useless with a single drive. The syntax looks like this:

```
COPY (Ddn,)("file1") TO (Ddn,)("file2")
```

To copy all files (for example, from drive 0 to drive 1), use the following command:

```
COPY D0 TO D1
```

## DCLOSE:

The command **DCLOSE** has the same function as the simple **CLOSE** command, with the following exceptions:

<b>DCLOSE</b>	closes all files
<b>DCLOSE#1</b>	closes file number 1
<b>DCLOSE#1 ON U9</b>	closes the logical file #1 on device address 9
<b>DCLOSE U8</b>	closes all files on device address 8

The command has the following syntax:

```
DCLOSE (#1fn) (ON Uda)
```

## DLOAD:

The command **DLOAD** has the advantage that the standard device address 8 used. The format:

```
DLOAD "program" (,Ddn)(,Uda)
```

For instance, if you want to load the program "PRG.2" from drive 0 or from a single drive, give the following command:

```
DLOAD "PRG.2"
```

Drive 0 (D0) is the default value.

## Anatomy of the 1541 Disk Drive

### DOPEN:

This command of BASIC 4.0 is very comprehensive. The following format verifies this:

```
DOPEN#lfn,"file"((,Ddn)((,Uda)((,fileparameter)
```

The peculiarity of this method of opening is the file parameter. There are two file parameters, that have the following function:

-----						
:	'L'-parameter	:	'W'-parameter	:	Mode of operation	:
-----						
:	YES	:	NO	:	A relative file is	:
:		:		:	opened.	:
:	NO	:	YES	:	A sequential file is	:
:		:		:	opened for writing.	:
:	NO	:	NO	:	A file is opened for	:
:		:		:	reading(REL,SEQ,PRG,USR):	:
-----						

In addition to the 'L' parameter the record length must be given (such as L80). A DOPEN command of this type looks like this:

```
DOPEN#1,"FILE.REL",D0,L80
```

Here a relative file is opened with a record length of 80 bytes. The declaration of the file parameter is only necessary once, at the establishment of the file. All later openings of the file can occur without the parameter declaration.

### DS\$ & DS:

After a disk error, the complete error message can be displayed with PRINT DS\$ or just the error number with PRINT DS. Of course, the error can be read within a program and the appropriate branch made. For example:

```
100 IF DS = 26 THEN GOTO ...
```

### DSAVE:

A program can be saved on disk with this command. The following format is to be noted:

```
DSAVE (Ddn,)"programname"((,Uda)
```

### HEADER:

A disk is formatted with the HEADER command in BASIC 4.0. It corresponds to the NEW command in BASIC 2.0. The syntax of the command:

```

or      HEADER "diskname",D0,Iid(U,da)
        HEADER Ddn,"diskname",Iid

```

Here there are two possibilities to designate the drive. The id is the diskette identification. If it is not given, the disk is presumed to be formatted and is merely given a new name and all files are erased.

#### RECORD:

This command corresponds to the position command of BASIC 2.0 (DOS 2.6). The read/write head can be positioned over a record in a relative file, without the need to send the position over channel 15. The syntax of this command illustrates how easy this positioning is:

```
RECORD#lfn,rn(,bp)
```

The logical file number is obtained from the opened relative file. 'rn' is the record number (1-65535) and 'bp' is the position within this record (1-254).

An example: You want to position the head over the twelfth byte of the 128th record of a relative file opened with the logical file number 2. The following command accomplishes this:

```
RECORD#2,128,12
```

#### RENAME:

This RENAME is similar to the RENAME of BASIC 2.0. The format of this command:

```
RENAME (Ddn,)"old name" TO "new name" (,Uda)
```

#### SCRATCH:

This method of erasing files is essentially easier because files can be erased with one command. The format of this command:

```
SCRATCH (Ddn,)"file" (,Uda)
```

After entering a SCRATCH command the message "ARE YOU SURE?" which allows the command to be stopped. If the file is really supposed to be erased, answer 'Y' else 'N'. After erasing the file, the message "FILES SCRATCHED" appears on the screen.



## Chapter 2: Advanced Disk Programming

### 2.1 Direct Access of any Block of the Diskette

When handling files and programs on the diskette, as described in Chapter 1, we didn't have to concern ourselves with the organization on the diskette, because the disk operating system (DOS) took care of these details for us.

But the DOS offers the capability of accessing each individual block on the diskette. This gives us a lot of flexibility - ranging from manipulation of individual files to creating completely new data structures.

In order to access a block directly, a channel is OPENed to a data buffer within the 1541 disk drive. It is over this channel that data is transmitted. The data buffer serves as an intermediate storage place for the data that is read from the diskette or written to the diskette. In order to inform the DOS that we want to work with direct access commands, we use a special filename in the OPEN command:

```
OPEN 1,8,2,"#"
```

Using this command, logical file number 1 on device 8 (the disk drive), is associated with a direct access file. Channel 2 serves to transmit data to and from the disk drive. The channel number (secondary address in the OPEN command) may be 2 through 14. Channels 0 and 1 are reserved for LOAD and SAVE and channel 15 is the command channel. The choice of a secondary address is arbitrary. You may not use the same secondary address simultaneously, since the DOS, upon encountering the second OPEN command with the same secondary address, closes the previous file using this channel number. This also occurs when working with sequential or relative files.

This form of the OPEN command causes the DOS to search for a free data buffer and assign it to that channel. By using a GET# statement immediately after the OPEN we can find the buffer number that the DOS assigns:

```
100 OPEN 1,8,2,"#"
110 GET#1, AS
120 PRINT ASC(AS+CHR$(0))
RUN
```

3

In this case, buffer three was assigned. The buffer numbers range from 0 to 4. Each buffer can hold 256 characters of data. The buffers are located in the following memory

locations in the VIC 1541:

Buffer number	Memory location
0	\$300-\$3FF, 768-1023
1	\$400-\$4FF, 1024-1279
2	\$500-\$5FF, 1280-1535
3	\$600-\$6FF, 1536-1791
4	\$700-\$7FF, 1792-2047

Buffer 4 is normally unavailable, because the BAM is stored there. If we work with sequential or relative files at the same time, buffer 3 is also unavailable, because it is used for the directory. If we want to associate a specific data buffer for direct access, we can assign it with the OPEN command.

### OPEN 1,8,2,"#3"

This associates buffer 3 (\$600-\$6FF) with channel number 2, assuming it is still free. Unless you have a pressing reason to use a specific buffer, you should leave the choice of the buffer up to the DOS, because the choice of a definite buffer increases the possibility that it will not be available.

After opening a channel, you should check the error channel.

```
130 OPEN 15,8,15
140 GET#15, A$ : PRINT A$; : IF ST<>64 THEN 140
```

If the buffer is already in use, you will receive the error message

```
70,NO CHANNEL,00,00
```

If no other files are open, you can open up to 4 channels for direct access. The following example illustrates this:

```
10 OPEN 1,8,15,"I0" : I=2 : REM ERROR CHANNEL
20 OPEN 2,8,2, "#" : GOSUB 100
30 OPEN 3,8,3, "#" : GOSUB 100
40 OPEN 4,8,4, "#" : GOSUB 100
50 OPEN 5,8,5, "#" : GOSUB 100
60 OPEN 6,8,6, "#" : GOSUB 100
70 END
100 GET#I,A$:PRINT ASC(A$+CHR$(0))
110 I=I+1 : REM BUFFER NUMBER
120 GET#1,A$ : PRINT A$; : IF ST<>64 THEN 120
130 RETURN
```

When RUN, the above program produces the following output:

## Anatomy of the 1541 Disk Drive

```
00, OK,00,00
2
00, OK,00,00
1
00, OK,00,00
0
00, OK,00,00
199
70,NO CHANNEL,00,00
```

As you see, attempting to open a fifth channel for direct access fails.

Transmitting data to and from the buffer usually takes place using the GET#, INPUT# and PRINT# statements.

If a buffer contains pure text (alphanumeric data) which is not longer than 88 characters and is separated using CR (Carriage Return, CHR\$(13)), it can be read using INPUT#. However, if the buffer contains control characters or the text is separated using commas or colons, the INPUT# statement fails. Then we must use the GET# statement, which retrieves only one character at a time. GET# does not allow null values (CHR\$(0)) to be read. In this case, GE1# receives an empty string and you must check for this condition as below:

```
100 GET#2, A$ : IF A$ + "" THEN A$ = CHR$(0)
```

A simpler alternative to the GET# statement is to use the statement INPUT\*, as is described in section 4.3.1. Here you can declare how many characters are to be read into a string. It also handles null values (CHR\$(0)). You can read almost the entire buffer (255 characters are possible) with one command.

In the next section, all commands used for direct access are described in detail. Keep the following points in mind when using direct access commands.

When using direct access commands, you must explicitly cause the blocks on the diskette to be read or written. The direct access commands are transmitted over command channel 15. The data that is read from or written to a buffer are transmitted over a separate channel that is associated with that buffer. Both channel 15 and the separate channel must be OPENed before transmission can begin.

- 1) A PRINT# statement to command channel 15, sends a direct access command to the DOS.
- 2) A PRINT# statement to channels 2 thru 14 sends data to a buffer.
- 3) An INPUT# or GET# statement to command channel 15 re-

turns any error messages detected by the DOS.

- 4) An INPUT# or GET# statement to channels 2 thru 14, reads the data from the buffer.

If you are ready to work with the block commands and want to display individual blocks on the screen or change them, you can use the DOS monitor in section 4.6, which provides a simple and easy way of doing so.

### 2.2 The Direct Access Commands

#### 2.2.1 The Block-Read Command B-R

The block-read command instructs the 1541 to read a block from the diskette into a buffer of a previously opened direct access file. The block-read command is sent over the command channel (secondary address 15) to the disk drive. The block-read command can be shortened to B-R. Because this command does not read the first byte of the block, you can substitute the command U1 to read a block. The command has the following syntax:

**U1 channelnumber drive track sector**

You must give the channel number that you used when OPENing the direct access file. Next follows the drive number, which is always zero for the VIC 1541, and then the track and sector numbers of the block you want to read.

```
10 OPEN 1,8,15
20 OPEN 2,8,2, "#"
30 PRINT#1, "U1 2 0 18 0"
```

This reads the contents of track 18 sector 0 into the buffer belonging to channel 2. Now you can read the data from this buffer with GET#2.

```
40 GET#2, AS,BS
50 PRINT ASC(AS), ASC(B$)

18      1
```

Now we have read and displayed the first two bytes in the buffer. Sector 0 of track 18 contains a pointer to the first directory block (track and sector) and the BAM for the diskette.

In the demo program DISPLAY T&S on the TEST/DEMO diskette (section 4.2.7) this command is used in order to read the BAM from the disk and to graphically display each record on the disk.

We can read all 256 bytes of the block from the buffer with the GET# statement; in our example we will read the diskette name and ID from position 144.

The blocks which comprise a file are chained to each other. The first two bytes of each file block contains a pointer to the track and sector of the following block. Using this information, you can piece together the usage of disk space for a file. A track pointer of zero indicates the last

block of the file and the pointer which usually contains the sector number now contains the number of bytes of the last block which are part of this file. The first sector of a file can be read with our program in section 4.1.1. The following small program displays all of the remaining tracks and sectors that are part of the file.

```

100 OPEN 1,8,15
110 OPEN 2,8,2, "#"
120 INPUT "TRACK AND SECTOR ";T,S
130 PRINT#1,"U1 2 0";T;S
140 GET#2, T$, S$
150 T = ASC(T$+CHR$(0)): S = ASC(S$+CHR$(0))
160 IF T=0 THEN CLOSE 2 : CLOSE 1 : END
170 PRINT "TRACK";T,"SECTOR";S
180 GOTO 130

```

Enter 18 and 0 as track and sector to follow the blocks for the BAM and directory.

### 2.2.2 The Block-Pointer Command B-P

The diskette name is located starting at position 144 of track 18, sector 0. Using the above example, we have to read the first 143 bytes of the buffer in order to be positioned at the diskette name. But the DOS has an easier way to do this. To access any desired byte of a buffer, you can use the block-pointer command. Using the block-pointer command the DOS moves to an exact position within the buffer. The block-pointer command can be shortened to B-P. The syntax is the following:

#### B-P channelnumber position

Now we can read the diskette name directly:

```

100 OPEN 1,8,15
110 OPEN 2,8,2, "#"
120 PRINT#1,"U1 2 0 18 0"
130 PRINT#1,"B-P 2 144"
140 FOR I = 1 TO 16 : REM MAXIMUM LENGTH
150 GET#2, A$ : IF A$=CHR$(160) THEN 170
160 PRINT A$; : NEXT
170 CLOSE 2 : CLOSE 1

```

Here we first read the block, set the buffer pointer to position 144 and then read and print the diskette name which has a maximum length of 16 characters. A shifted space (CHR\$(160)) indicates the end of the diskette name.

The bytes in the buffer are numbered 0 through 255, the first byte having the number 0. The buffer pointer is auto-

## Anatomy of the 1541 Disk Drive

matically set to zero by reading a block with U1. You can, for example, read byte number 2 after reading the name. You do this by setting the buffer pointer to this value.

```
PRINT#1, "B-P 2 2"
```

### 2.2.3 The Block-Write Command B-W

The block-write command allows us to write the contents of a buffer to a desired block on the diskette. With this, you can write the block one has sent to the buffer within the disk drive.

It is possible to read a block into the buffer with the block-read command, change some bytes, and then write the block back. The block-write command can be shortened to B-W. Because this B-W command writes the contents of the buffer pointer, one usually uses the U2 command which always sets the buffer pointer to 1. The syntax of the command is analogous to the B-R command:

**U2 channelnumber drive track sector**

```
100 OPEN 1,8,15
110 OPEN 2,8,2, "#"
120 PRINT#2, "TEST DATA"
130 PRINT#1, "U2 2 0 1 0"
140 CLOSE 2 : CLOSE 1
```

Here the text "TEST DATA" will be written to the buffer associated to channel 2 and then written to track 1 sector 0 of the diskette. The U2 command does not change the contents of the buffer.

Here's an example of using the block-write command to change the diskette name that we read in the last section. For this we must fill the new name with 16 characters ending with a shifted spaces CHR\$(160), so that we can write it to the disk. We will again use the block-pointer command to set the buffer pointer directly to the desired position within the buffer.

```
100 OPEN 1,8,15
110 OPEN 2,8,2, "#"
120 PRINT#1, "U1 2 0 18 0"
130 PRINT#1, "B-P 2 144"
140 AS="NEW FILE NAME"
150 IF LEN(AS)<16 THEN AS=AS+CHR$(160) : GOTO 150
160 PRINT#2, AS;
170 PRINT#1, "U2 2 0 18 0"
180 CLOSE 2
190 PRINT#1, "I0" : CLOSE 1
```

First we read track 18 sector 0 into the buffer, set the buffer pointer to the position of the diskette name and write a new 16 character name to the buffer. Note that the diskette name is changed in the buffer only. But in line 170, the buffer contents are written to the same block which changes the name permanently on the diskette. Next channel 2 is closed. Finally the diskette is initialized so the BAM and name in the DOS memory are updated. Get the directory with

```
LOAD"$",8
LIST
```

on the screen to verify that the diskette name has changed.

## 2.2.4 The Block-Allocate Command B-A

The block-allocate command has the task of indicating in the BAM (block availability map) is a particular diskette block is being used. The block allocate command can be shortened to **B-A**. For program, sequential or relative files, as diskette blocks are used, the BAM is updated to note that the block is no longer available. But blocks written using the direct access commands are not automatically allocated. When blocks used in this manner are not allocated, the possibility exists that they will be overwritten when other files are used. The block-allocate command can be used to prevent this overwriting. The block-allocate command has the following syntax:

### B-A drive track sector

With this the corresponding block in the BAM is marked as allocated and is protected from being overwritten by other files. If the block was already allocated, the error channel returns error message 65, 'NO BLOCK'.

```
100 OPEN 1,8,15
110 INPUT "TRACK, SECTOR ";T,S
120 PRINT#1, "B-A 0";T,S
130 INPUT#1, A$,B$,C$,D$
140 PRINT A$,"B$","C$","D$"
```

Using this program you can input a track and sector number of a block that you want to allocate. If the block is still free, it was allocated and the message **00, OK,00,00** is returned. If that block is already allocated, the message **65,NO BLOCK,TT,SS** is returned. In this case TT and SS contain the next higher numbered free block on the diskette. This tells you that the requested block is allocated but the block at TT,SS is still available. If error message 65 returns zeroes as the track and sector numbers, it means



## Anatomy of the 1541 Disk Drive

that no block with a higher track and/or sector number is available. The following program automatically allocates the next free sector:

```
100 OPEN 1,8,15
110 INPUT "TRACK, SECTOR ";T,S
120 PRINT#1, "B-A 0";T;S
130 INPUT#1, A$,B$,TT,SS
140 IF A$ = "00" THEN 190
150 IF A$<>"65" THEN PRINT A$,"B$","TT","SS : END
160 IF TT=0 THEN PRINT "NO MORE FREE BLOCKS" : END
170 IF TT=18 THEN TT=19 : SS=0
180 T=TT : S=SS : GOTO 120
190 PRINT "TRACK" TT "SECTOR" SS "ALLOCATED."
```

The test for track 18 in line 180 prevents a block in the directory from being allocated. An additional error message in connection with the B-A command is interesting. If one attempts to allocate a block that does not exist, for example, track 20 sector 21, one received the error message

### 66,ILLEGAL TRACK OR SECTOR,20,21

Marking a block as allocated in the BAM prevents it from being overwritten by other files. The block will be recognized as allocated until the command **VALIDATE** (COLLECT in BASIC 4.0) is issued. The **VALIDATE** command rebuilds a new BAM by rechainning the blocks of individual files and marking each block as belonging to a new BAM. Unclosed files, marked in the directory with \* are deleted. All blocks allocated with the B-A command and those not belonging to a properly closed file are freed. So, if you allocate blocks that do not belong to a file that appears in the directory, you should not use the **VALIDATE** command, or the blocks will be freed, thus destroying your file.

### 2.2.5 The Block-Free Command B-F

The block-free command performs the opposite function of the block-allocate command. It marks a block as not allocated (free) in the BAM. The block-free command can be shortened to **B-F**. The syntax is analogous to the block-allocate command:

#### B-F drive track sector

```
100 OPEN 1,8,15
110 PRINT#1, "B-F 0 20 9"
```

Here the block in track 20 sector 9 is freed in the BAM. If this block is already free, no error occurs.

Allocating and freeing blocks has an effect only on the blocks used by program, sequential or relative file by the DOS. The block-write and block-read commands do not check the BAM before overwriting blocks. With these commands you can write to blocks marked as allocated in the BAM. If, for example, you have a disk containing only direct access files, it is in principle unnecessary to allocate written blocks because no other files will be written on the diskette. In this case, you can use the directory blocks in track 18 and have 672 blocks available on the VIC 1541 diskette.

### 2.2.6 The Block-Execute Command B-E

The block-execute command allows a block to be read from diskette into a buffer and then the contents of the buffer to be executed as a machine language program. You can write routines that the DOS is supposed to execute with the B-W or U2 command to a sector and later load it into a buffer with the block-execute program where it will be executed as a machine language program. Naturally, this presupposes knowledge of the internal workings of the DOS. If you want to use the B-E command, you usually give the buffer number in the OPEN command, in case the machine language program is not relocatable and is written for a specific buffer. The block-execute command has the following syntax:

**B-E channelnumber drive track sector**

```
100 OPEN 1,8,15
110 OPEN 2,8,2, "#3"
120 PRINT#1, "B-E 2 0 17 12"
```

Here buffer 3 (\$600-\$6FF) is assigned to channel 2. The contents of track 17 sector 12 is loaded into this buffer and there the machine language program is executed.

The block-execute command is a combination of the block-read and memory-execute commands. Examples of the design of machine language programs to execute in the DOS are found in section 2.4 by the memory commands.

## Anatomy of the 1541 Disk Drive

### 2.3 Uses of direct access

What do the direct access commands permit us to do?

Here is a sample of their use:

By manipulating individual sectors you can make changes to the BAM sector (Track 18, Sector 0) such as changing the diskette name or ID.

You can make changes to the DIRECTORY (beginning at Track 18, Sector 1). Each file entry in the directory has unused space. You can use the unused space to store additional information.

You can change file names in the directory by using direct access commands.

You can follow the "chaining" of the blocks in a file to determine if the file is intact.

You can CLOSE an unclosed file by setting bit 7 of the file type indicator in the directory. For example, you can change the file type indicator from \$02 to \$82. Normally these files are indicated in the directory with an asterisk; after the above change the asterisk will disappear.

Each file entry also contains a "lock" which disallows deletion (SCRATCH command). If you set bit 6 of the file type then the file is said to be locked and not available for deletion. These entries have the < symbol after the type designation in the directory listing. Using this bit of knowledge, you can protect important programs on your diskette from accidental erasure. More information on this topic is found in section 4.1.

If you are interested in making such changes, you may want to read an entire sector and display it on the screen, change it, and write it back again. Such a program called the DISK MONITOR is described in section 4.6. Before you begin with such experiments, however, you should make a copy of your diskette. A directory or BAM error can result in the loss of the entire diskette contents.

Have you ever accidentally scratched a program or file from a diskette? As long as you haven't written any other programs or data to the diskette, you can recover this scratched file. Scratching a file simply sets the file type to 0 in the directory and frees the allocated blocks. You need only search the directory entries for the file and restore the file type: \$81 for SEQ, \$82 for PRG, \$83 for USR, and \$84 for REL. After restoring the file type, you should use the VALIDATE command to reallocate the blocks again (for example: OPEN 1,8,15:PRINT#1,"V0").

Other uses of direct access can provide the means for creating new data structures that the DOS normally does not recognize. You can undertake the management of the new file yourself, and use the direct access commands for reading and writing. Such a data structure is the ISAM file. ISAM is an abbreviation for Indexed Sequential Access Method. With an ISAM file, you can directly access each record, similar to the relative file. However, access is not by the record number, however, but by a **key** or **index**. This index is a field within the record. If, for example, a record consists of 5 fields, last name, first name, street, city/state and zip code, last name can be defined as the access key. To read the record **Muller**, the command is simply 'read record "Muller"'. We need not concern ourselves with record number or other ordering criteria and can select which record we want to read, change, write or erase with clear text. In such an ISAM file system, the index is usually saved separately, together with the information where the data record can be found on the disk. Such an ISAM file management with very powerful additions as described here, is found along with other features in the program development system **MASTER 64**, also available for the Commodore 64 from Abacus Software.

### 2.4 Accessing the DOS - The Memory Commands

In section 2.2.6 we saw a way to load a program into DOS memory and execute it. With the memory commands, we can access each byte of the DOS and execute programs in RAM and ROM. For instance, we can access the work space of the DOS and read the number of free blocks on the disk or get the disk name from the BAM buffer. By writing into the DOS RAM we can change constants such as the device number of the drive or the number of read attempts for a block until an error message results. Furthermore, we can execute routines inside the DOS memory. These can be DOS ROM routines or your own, that are stored in a buffer and executes there. Of course this presumes knowledge of 6502 machine language and of the method of operation of the DOS. We hope this book is be helpful for the latter. Now follows a description of the commands and examples of their use.

#### 2.4.1 The Memory-Read Command M-R

Using this command, you can access each byte of the DOS. The memory-read command can be shortened to M-R. The memory-read command is transmitted over the command channel. The byte read is then returned over the command channel where it can be retrieved with GET#. The syntax of the command looks like this:

**M-R CHR\$(LO) CHR\$(HI)**

LO and HI signify the low and high bytes of the address in the DOS that should be read. The following program asks for an address and reads the contents of the address out of the DOS.

```
100 INPUT"ADDRESS ";A
110 HI = INT (A/256)
120 LO = A-256*HI
130 OPEN 1,8,15
140 PRINT#1, "M-R";CHR$(LO);CHR$(HI)
150 GET#1,A$
160 PRINT ASC(A$+CHR$(0))
```

For instance, if we want to know the number of free blocks on a diskette, we don't have to read the entire directory, rather we can read the appropriate bytes directly from the DOS storage. This may be necessary if files are to be established by a program and you don't know if there is enough space on the disk.

```
100 OPEN 1,8,15,"IO"
110 PRINT#1, "M-R" CHR$(250) CHR$(2)
120 GET#1, A$ : IF A$="" THEN A$=CHR$(0)
```

```

130 PRINT#1, "M-R" CHR$(252) CHR$(2)
140 GET#1, B$ : IF B$="" THEN B$=CHR$(0)
150 PRINT ASC(A$) + 256 * ASC(B$) "BLOCKS FREE"
160 CLOSE 1

```

With this syntax, an M-R command must be given for each byte that is to be read. As you can gather from the DOS listing and through checking and verifying, one can read more than one byte at a time with a M-R command. You need only give the number of bytes to be read as the third parameter:

#### **M-R CHR\$(LO) CHR\$(HI) CHR\$(NUMBER)**

We can use this to read the name of a diskette from the BAM buffer storage. Before this can be done, the diskette must be initialized so that the current diskette name is stored in the buffer at address \$700, out of which we will read the name of the disk with the M-R command.

```

100 OPEN 1,8,15, "IO"
110 PRINT#1, "M-R" CHR$(144) CHR$(7) CHR$(16)
120 INPUT#1, A$
130 PRINT A$

```

This is a simple way to read the name of the diskette (16 characters padded with shifted spaces (CHR\$(160))). With this you can check if the correct diskette is in the drive.

The disk buffer can also be read using this method. It also allows parts of the DOS to be manipulated by copying the contents of the ROM to a buffer where it can be changed and executed. This is explained in the next two sections.

### **2.4.2 The Memory-Write Command M-W**

The complement command of memory-read is the command to write data in the DOS storage memory-write or M-W. Writing is allowed only to DOS RAM - page zero, stack, and buffers. It is possible to send several bytes with one command. The syntax look like this:

**M-W CHR\$(LO) CHR\$(HI) CHR\$(NUMBER) CHR\$(DATA1) CHR\$(DATA2)**

The number of bytes as specified by NUMBER can be transmitted, theoretically 255, but because the input buffer holds only 40 characters, the number of bytes is limited to 34. A possible use of this command is to change the address number (see program 'DISK ADDRESS CHANGE', section 4.2.3). The address is stored in two memory locations in page zero. The device number plus \$20 (32 decimal) is stored in address \$77 (119 decimal) for LISTEN, for receiving data from the computer. The address immediately following contains the

## Anatomy of the 1541 Disk Drive

device number plus \$40 (64 decimal) for TALK, for sending data to the computer. Because the addresses are saved separately. It is possible to use different send and receive addresses. In the following example, the receive address is set to 9 and the send address to 10.

```
100 OPEN 1,8,15
110 PRINT#1, "M-W" CHR$(119) CHR$(0) CHR$(2)
      CHR$(9+32) CHR$(10+64)
120 CLOSE 1
140 OPEN 1,9,15
150 OPEN 2,10,15
160 PRINT#1,"IO"
170 INPUT#2,AS,BS,CS,DS
180 PRINT AS","BS","CS","DS

00, OK,00,00
```

Programs cannot be loaded this way because the DOS will try to load the program using the same address that the filename was sent under.

Changing the device number is necessary if you want to use more than one disk drive with a single computer. To this end, change the device address of the second drive to 9. This software change remains in effect only until a reset (for example, turning the drive off). If the change needs to be permanent, you can change the with DIP switches or cut the circuit board jumper inside the drive.

Because many parameters of the DOS are in RAM, you can make extensive changes to the function of the DOS, such as the step size, with which the number of sectors per track is determined (address \$69 (105 decimal), normally contains 10). We can also specify the number of attempted reads until an error results (address \$6A (106 decimal), contains 5). More addresses of parameters can be found in section 3.1.2.

### 2.4.3 The Memory-Execute Command M-E

Using this command you can call up and execute machine language programs in the DOS memory. The memory-execute command can be shortened to M-E. The programs must end with RTS (Return from Subroutine, \$60). The syntax of the command:

**M-E CHR\$(LO) CHR\$(HI)**

Again, LO and HI are the low and high bytes of the starting address of the machine language routine. It is possible to call up routines in the DOS ROM as well as our own routines written to a buffer with M-W and there executed. As an

example, you can call up a routine that creates an error message. For example, address \$EFC9 is the entry point for message 72, 'DISK FULL'. The example looks like this:

```
100 OPEN 1,8,15
110 PRINT#1,"M-E" CHR$(201) CHR$(239)
120 INPUT#1,A$,B$,C$,D$
130 PRINT AS "," B$ "," C$ "," D$
```

In line 110, the address \$EFC9 is divided into a low byte of \$C9 (201) and high byte of \$EF (239) and sent as the parameters of the M-E command. Then the error channel is read and the message displayed.

## 72,DISK FULL,00,00

If you want to run your own programs in the 1541 drive, the program should be written to a buffer and there called with M-E. Should this program be used more often, the contents of the buffer can be written to a block on the diskette. It can then be executed with the B-E command, which loads the contents of the block in the buffer and then automatically starts the routine. As a suggestion for your own program in DOS, you can display the directory in a different form, with additional parameters, similar to the program in section 4.1.1. In addition, you could count the number of files on the disk and display that. Using such a routine you can get a much clearer understanding of how the directory is created in the DOS listing. If you are clear on the matter of the new directory format, you are ready to take the additional parameters from the directory entries and assemble them in the desired format.

### 2.4.4 The User Commands U

Using the USER commands there are two possible ways of executing programs in the drive. The user commands have the following syntax:

UX

X can be a letter from A to J or a digit from 1 to 9 or ':' (which takes the place of 10). When a command is called, a jump is made to the following addresses in DOS:

UA	U1	\$CD5F	substitute for 'Block-Read'
UB	U2	\$DC97	substitute for 'Block-Write'
UC	U3	\$0500	
UD	U4	\$0503	
UE	U5	\$0506	
UF	U6	\$0509	
UG	U7	\$050C	



## Anatomy of the 1541 Disk Drive

UH	U8	\$050F	
UI	U9	\$FF01	
UJ	U:	\$EAA0	reset

You are already acquainted with the commands U1 and U2 (also UA and UB); they serve as substitutes for BLOCK-READ and BLOCK-WRITE. The commands U3 to U8 (UC to UH) jump to addresses within buffer 2 (address \$500 (1280) - see section 2.1). If you want to use several commands, a jump table to individual routines can be placed there; if only one user command (U3) is used, the program can begin directly at \$500.

The user command UJ jumps to the reset vector; the disk drive is then reset.

```
100 OPEN 1,8,15
110 PRINT#1,"UJ"
120 FOR I=1 TO 1000 : NEXT
130 GET#1,A$ : PRINT A$ : IF ST<>64 THEN 130
```

```
73,CBM DOS V2.6 1541,00,00
```

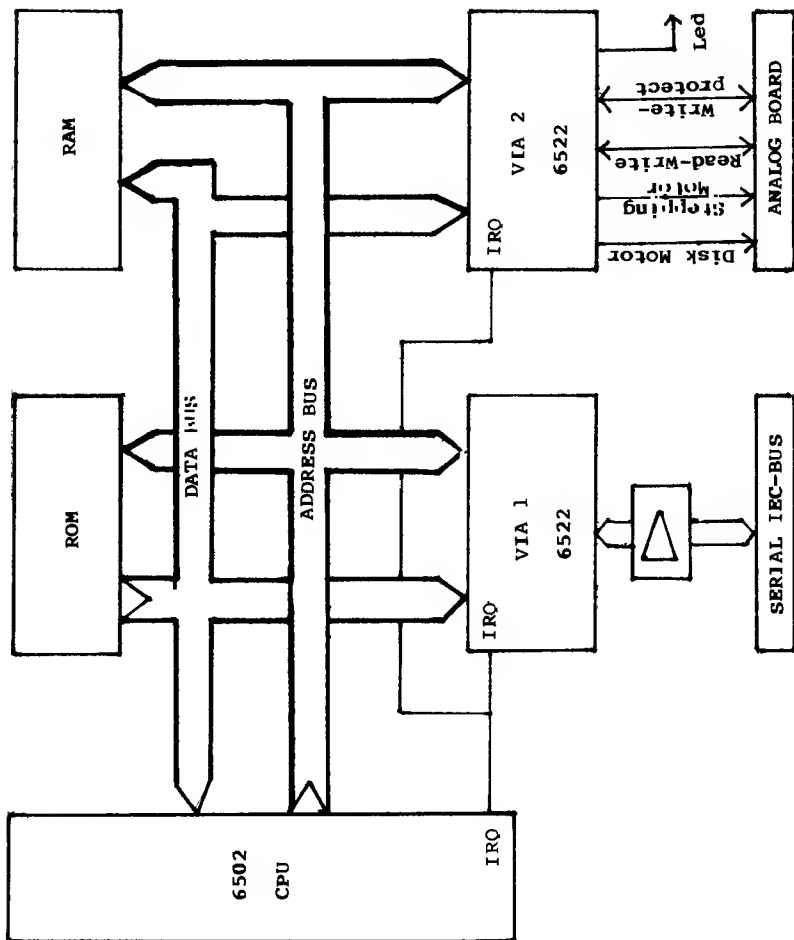
Line 120 waits for the reset to take place. Then the initialization message is retrieved in line 130.

By using the user commands, parameters can be passed to the routines. The complete command string is put in the input buffer at \$200 (512). Possible parameters are addresses, command codes, and filenames. This way, the user commands can be utilized to expand the commands of the disk or to realize a new data structure. Whole user commands can replace the M-E command with its corresponding addresses; the user-call is shorter and clearer.

## Chapter 3: Technical Information

### 3.1 The Construction of the VIC 1541

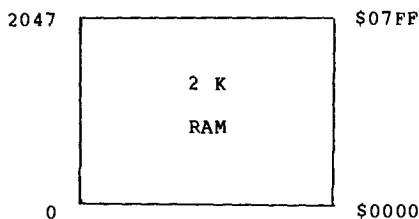
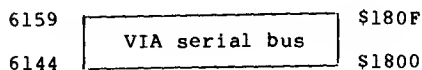
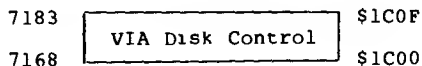
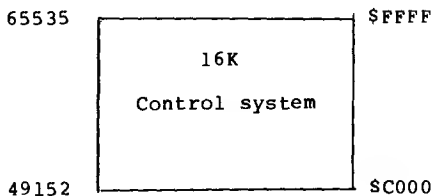
#### 3.1.1 Block Diagram of the Disk Drive



## Anatomy of the 1541 Disk Drive

### 3.1.2 DOS Memory Map - ROM, RAM, I/O

Memory map of the VIC 1541 disk drive



Allocating and freeing blocks has an effect only on the blocks used by program, sequential or relative file by the DOS. The block-write and block-read commands do not check the BAM before overwriting blocks. With these commands you can write to blocks marked as allocated in the BAM. If, for example, you have a disk containing only direct access files, it is in principle unnecessary to allocate written blocks because no other files will be written on the diskette. In this case, you can use the directory blocks in track 18 and have 672 blocks available on the VIC 1541 diskette.

### 2.2.6 The Block-Execute Command B-E

The block-execute command allows a block to be read from diskette into a buffer and then the contents of the buffer to be executed as a machine language program. You can write routines that the DOS is supposed to execute with the B-W or U2 command to a sector and later load it into a buffer with the block-execute program where it will be executed as a machine language program. Naturally, this presupposes knowledge of the internal workings of the DOS. If you want to use the B-E command, you usually give the buffer number in the OPEN command, in case the machine language program is not relocatable and is written for a specific buffer. The block-execute command has the following syntax:

**B-E channelnumber drive track sector**

```
100 OPEN 1,8,15
110 OPEN 2,8,2, "#3"
120 PRINT#1, "B-E 2 0 17 12"
```

Here buffer 3 (\$600-\$6FF) is assigned to channel 2. The contents of track 17 sector 12 is loaded into this buffer and there the machine language program is executed.

The block-execute command is a combination of the block-read and memory-execute commands. Examples of the design of machine language programs to execute in the DOS are found in section 2.4 by the memory commands.

## Anatomy of the 1541 Disk Drive

### 2.3 Uses of direct access

What do the direct access commands permit us to do?

Here is a sample of their use:

By manipulating individual sectors you can make changes to the BAM sector (Track 18, Sector 0) such as changing the diskette name or ID.

You can make changes to the DIRECTORY (beginning at Track 18, Sector 1). Each file entry in the directory has unused space. You can use the unused space to store additional information.

You can change file names in the directory by using direct access commands.

You can follow the "chaining" of the blocks in a file to determine if the file is intact.

You can CLOSE an unclosed file by setting bit 7 of the file type indicator in the directory. For example, you can change the file type indicator from \$02 to \$82. Normally these files are indicated in the directory with an asterisk; after the above change the asterisk will disappear.

Each file entry also contains a "lock" which disallows deletion (SCRATCH command). If you set bit 6 of the file type then the file is said to be locked and not available for deletion. These entries have the < symbol after the type designation in the directory listing. Using this bit of knowledge, you can protect important programs on your diskette from accidental erasure. More information on this topic is found in section 4.1.

If you are interested in making such changes, you may want to read an entire sector and display it on the screen, change it, and write it back again. Such a program called the DISK MONITOR is described in section 4.6. Before you begin with such experiments, however, you should make a copy of your diskette. A directory or BAM error can result in the loss of the entire diskette contents.

Have you ever accidentally scratched a program or file from a diskette? As long as you haven't written any other programs or data to the diskette, you can recover this scratched file. Scratching a file simply sets the file type to 0 in the directory and frees the allocated blocks. You need only search the directory entries for the file and restore the file type: \$81 for SEQ, \$82 for PRG, \$83 for USR, and \$84 for REL. After restoring the file type, you should use the VALIDATE command to reallocate the blocks again (for example: OPEN 1,8,15:PRINT#1,"V0").

Other uses of direct access can provide the means for creating new data structures that the DOS normally does not recognize. You can undertake the management of the new file yourself, and use the direct access commands for reading and writing. Such a data structure is the ISAM file. ISAM is an abbreviation for Indexed Sequential Access Method. With an ISAM file, you can directly access each record, similar to the relative file. However, access is not by the record number, however, but by a **key** or **index**. This index is a field within the record. If, for example, a record consists of 5 fields, last name, first name, street, city/state and zip code, last name can be defined as the access key. To read the record **Muller**, the command is simply 'read record "Muller"'. We need not concern ourselves with record number or other ordering criteria and can select which record we want to read, change, write or erase with clear text. In such an ISAM file system, the index is usually saved separately, together with the information where the data record can be found on the disk. Such an ISAM file management with very powerful additions as described here, is found along with other features in the program development system **MASTER 64**, also available for the Commodore 64 from Abacus Software.

### 2.4 Accessing the DOS - The Memory Commands

In section 2.2.6 we saw a way to load a program into DOS memory and execute it. With the memory commands, we can access each byte of the DOS and execute programs in RAM and ROM. For instance, we can access the work space of the DOS and read the number of free blocks on the disk or get the disk name from the BAM buffer. By writing into the DOS RAM we can change constants such as the device number of the drive or the number of read attempts for a block until an error message results. Furthermore, we can execute routines inside the DOS memory. These can be DOS ROM routines or your own, that are stored in a buffer and executes there. Of course this presumes knowledge of 6502 machine language and of the method of operation of the DOS. We hope this book is be helpful for the latter. Now follows a description of the commands and examples of their use.

#### 2.4.1 The Memory-Read Command M-R

Using this command, you can access each byte of the DOS. The memory-read command can be shortened to M-R. The memory-read command is transmitted over the command channel. The byte read is then returned over the command channel where it can be retrieved with GET#. The syntax of the command looks like this:

**M-R CHR\$(LO) CHR\$(HI)**

LO and HI signify the low and high bytes of the address in the DOS that should be read. The following program asks for an address and reads the contents of the address out of the DOS.

```
100 INPUT"ADDRESS ";A
110 HI = INT (A/256)
120 LO = A-256*HI
130 OPEN 1,8,15
140 PRINT#1, "M-R";CHR$(LO);CHR$(HI)
150 GET#1,A$
160 PRINT ASC(A$+CHR$(0))
```

For instance, if we want to know the number of free blocks on a diskette, we don't have to read the entire directory, rather we can read the appropriate bytes directly from the DOS storage. This may be necessary if files are to be established by a program and you don't know if there is enough space on the disk.

```
100 OPEN 1,8,15,"IO"
110 PRINT#1, "M-R" CHR$(250) CHR$(2)
120 GET#1, A$ : IF A$="" THEN A$=CHR$(0)
```

```

130 PRINT#1, "M-R" CHR$(252) CHR$(2)
140 GET#1, B$ : IF B$="" THEN B$=CHR$(0)
150 PRINT ASC(A$) + 256 * ASC(B$) "BLOCKS FREE"
160 CLOSE 1

```

With this syntax, an M-R command must be given for each byte that is to be read. As you can gather from the DOS listing and through checking and verifying, one can read more than one byte at a time with a M-R command. You need only give the number of bytes to be read as the third parameter:

#### **M-R CHR\$(LO) CHR\$(HI) CHR\$(NUMBER)**

We can use this to read the name of a diskette from the BAM buffer storage. Before this can be done, the diskette must be initialized so that the current diskette name is stored in the buffer at address \$700, out of which we will read the name of the disk with the M-R command.

```

100 OPEN 1,8,15, "IO"
110 PRINT#1, "M-R" CHR$(144) CHR$(7) CHR$(16)
120 INPUT#1, A$
130 PRINT A$

```

This is a simple way to read the name of the diskette (16 characters padded with shifted spaces (CHR\$(160))). With this you can check if the correct diskette is in the drive.

The disk buffer can also be read using this method. It also allows parts of the DOS to be manipulated by copying the contents of the ROM to a buffer where it can be changed and executed. This is explained in the next two sections.

### **2.4.2 The Memory-Write Command M-W**

The complement command of memory-read is the command to write data in the DOS storage memory-write or M-W. Writing is allowed only to DOS RAM - page zero, stack, and buffers. It is possible to send several bytes with one command. The syntax look like this:

**M-W CHR\$(LO) CHR\$(HI) CHR\$(NUMBER) CHR\$(DATA1) CHR\$(DATA2)**

The number of bytes as specified by NUMBER can be transmitted, theoretically 255, but because the input buffer holds only 40 characters, the number of bytes is limited to 34. A possible use of this command is to change the address number (see program 'DISK ADDRESS CHANGE', section 4.2.3). The address is stored in two memory locations in page zero. The device number plus \$20 (32 decimal) is stored in address \$77 (119 decimal) for LISTEN, for receiving data from the computer. The address immediately following contains the



## Anatomy of the 1541 Disk Drive

device number plus \$40 (64 decimal) for TALK, for sending data to the computer. Because the addresses are saved separately. It is possible to use different send and receive addresses. In the following example, the receive address is set to 9 and the send address to 10.

```
100 OPEN 1,8,15
110 PRINT#1, "M-W" CHR$(119) CHR$(0) CHR$(2)
      CHR$(9+32) CHR$(10+64)
120 CLOSE 1
140 OPEN 1,9,15
150 OPEN 2,10,15
160 PRINT#1,"IO"
170 INPUT#2,A$,B$,C$,D$
180 PRINT A$,"B$","C$","D$

00, OK,00,00
```

Programs cannot be loaded this way because the DOS will try to load the program using the same address that the filename was sent under.

Changing the device number is necessary if you want to use more than one disk drive with a single computer. To this end, change the device address of the second drive to 9. This software change remains in effect only until a reset (for example, turning the drive off). If the change needs to be permanent, you can change the with DIP switches or cut the circuit board jumper inside the drive.

Because many parameters of the DOS are in RAM, you can make extensive changes to the function of the DOS, such as the step size, with which the number of sectors per track is determined (address \$69 (105 decimal), normally contains 10). We can also specify the number of attempted reads until an error results (address \$6A (106 decimal), contains 5). More addresses of parameters can be found in section 3.1.2.

### 2.4.3 The Memory-Execute Command M-E

Using this command you can call up and execute machine language programs in the DOS memory. The memory-execute command can be shortened to M-E. The programs must end with RTS (Return from Subroutine, \$60). The syntax of the command:

**M-E CHR\$(LO) CHR\$(HI)**

Again, LO and HI are the low and high bytes of the starting address of the machine language routine. It is possible to call up routines in the DOS ROM as well as our own routines written to a buffer with M-W and there executed. As an

example, you can call up a routine that creates an error message. For example, address \$EFC9 is the entry point for message 72, 'DISK FULL'. The example looks like this:

```
100 OPEN 1,8,15
110 PRINT#1,"M-E" CHR$(201) CHR$(239)
120 INPUT#1,A$,B$,C$,D$
130 PRINT AS "," B$ "," C$ "," D$
```

In line 110, the address \$EFC9 is divided into a low byte of \$C9 (201) and high byte of \$EF (239) and sent as the parameters of the M-E command. Then the error channel is read and the message displayed.

## 72,DISK FULL,00,00

If you want to run your own programs in the 1541 drive, the program should be written to a buffer and there called with M-E. Should this program be used more often, the contents of the buffer can be written to a block on the diskette. It can then be executed with the B-E command, which loads the contents of the block in the buffer and then automatically starts the routine. As a suggestion for your own program in DOS, you can display the directory in a different form, with additional parameters, similar to the program in section 4.1.1. In addition, you could count the number of files on the disk and display that. Using such a routine you can get a much clearer understanding of how the directory is created in the DOS listing. If you are clear on the matter of the new directory format, you are ready to take the additional parameters from the directory entries and assemble them in the desired format.

### 2.4.4 The User Commands U

Using the USER commands there are two possible ways of executing programs in the drive. The user commands have the following syntax:

UX

X can be a letter from A to J or a digit from 1 to 9 or ':' (which takes the place of 10). When a command is called, a jump is made to the following addresses in DOS:

UA	U1	\$CD5F	substitute for 'Block-Read'
UB	U2	\$DC97	substitute for 'Block-Write'
UC	U3	\$0500	
UD	U4	\$0503	
UE	U5	\$0506	
UF	U6	\$0509	
UG	U7	\$050C	

## Anatomy of the 1541 Disk Drive

UH	U8	\$050F	
UI	U9	\$FF01	
UJ	U:	\$EAA0	reset

You are already acquainted with the commands U1 and U2 (also UA and UB); they serve as substitutes for BLOCK-READ and BLOCK-WRITE. The commands U3 to U8 (UC to UH) jump to addresses within buffer 2 (address \$500 (1280) - see section 2.1). If you want to use several commands, a jump table to individual routines can be placed there; if only one user command (U3) is used, the program can begin directly at \$500.

The user command UJ jumps to the reset vector; the disk drive is then reset.

```
100 OPEN 1,8,15
110 PRINT#1,"UJ"
120 FOR I=1 TO 1000 : NEXT
130 GET#1,A$ : PRINT A$ : IF ST<>64 THEN 130
```

```
73,CBM DOS V2.6 1541,00,00
```

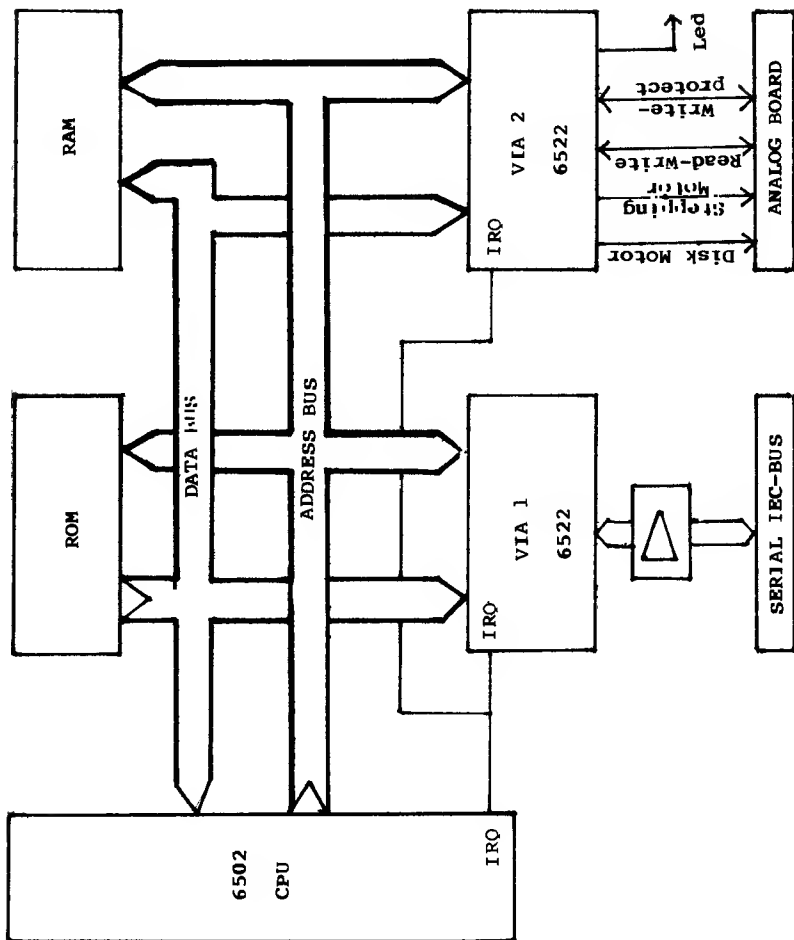
Line 120 waits for the reset to take place. Then the initialization message is retrieved in line 130.

By using the user commands, parameters can be passed to the routines. The complete command string is put in the input buffer at \$200 (512). Possible parameters are addresses, command codes, and filenames. This way, the user commands can be utilized to expand the commands of the disk or to realize a new data structure. Whole user commands can replace the M-E command with its corresponding addresses; the user-call is shorter and clearer.

## Chapter 3: Technical Information

### 3.1 The Construction of the VIC 1541

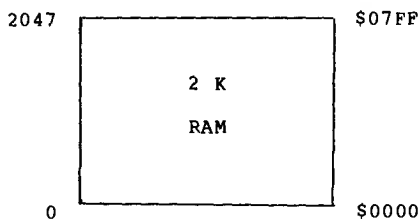
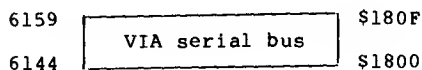
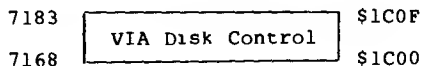
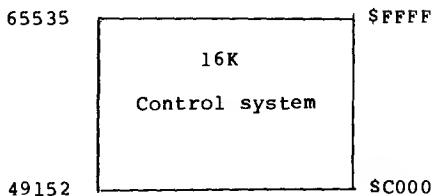
#### 3.1.1 Block Diagram of the Disk Drive



## Anatomy of the 1541 Disk Drive

### 3.1.2 DOS Memory Map - ROM, RAM, I/O

Memory map of the VIC 1541 disk drive



## Layout of the I/O Ports (VIA 6522)

VIA 6522 1, Port for Serial Bus

\$1800	Port B
\$1801	Port A
\$1802	Direction of Port B
\$1803	Direction of Port A
\$1805	Timer
PB 0:	DATA IN
PB 1:	DATA OUT
PB 2:	CLOCK IN
PB 3:	CLOCK OUT
PB 4:	ATN A
PB 5,6:	Device address
CB 2:	ATN IN

VIA 6522 2, Port for Motor and Read/Write Head Control

\$1C00	Port B, control port
\$1C01	Port A, data to and from read/write head
\$1C02	Direction of Port A
\$1C03	Direction of Port B
PB 0:	STP I
PB 1:	STP O      step motor for head movement
PB 2:	MTR      drive motor
PB 3:	ACT      LED on drive
PB 4:	WPS      Write Protect Switch
PB 7:	SYNC
CA 1:	Byte ready
CA 2:	SOE

## Anatomy of the 1541 Disk Drive

### The Layout of the Important Memory Locations

0	\$00	Command code for buffer 0
1	\$01	Command code for buffer 1
2	\$02	Command code for buffer 2
3	\$03	Command code for buffer 3
4	\$04	Command code for buffer 4
6	\$06-\$07	Track and sector for buffer 0
8	\$08-\$09	Track and sector for buffer 1
10	\$0A-\$0B	Track and sector for buffer 2
12	\$0C-\$0D	Track and sector for buffer 3
14	\$0E-\$0F	Track and sector for buffer 4
18	\$12-\$13	ID for drive 0
20	\$14-\$15	ID for drive 1
22	\$16-\$17	ID
32	\$20-\$21	Flag for head transport
48	\$30-\$31	Buffer pointer for disk controller
57	\$39	Constant 8, mark for beginning of data block header
58	\$3A	Parity for data buffer
61	\$3D	Drive number for disk controller
63	\$3F	Buffer number for disk controller
67	\$43	Number of sectors per track for formatting
71	\$47	Constant 7, mark for beginning of data block header
73	\$49	Stack pointer
74	\$4A	Step counter for head transport
81	\$51	Actual track number for formatting
105	\$69	Step size for sector division (10)
106	\$6A	Number of read attempts (5)
111	\$6F-\$70	Pointer to address for M & B commands
119	\$77	Device number + \$20 for listen
120	\$78	Device number + \$40 for talk
121	\$79	Flag for listen (1/0)
122	\$7A	Flag for talk (1/0)
124	\$7C	Flag for ATN from serial bus receiving
125	\$7D	Flag for EOI from serial bus
127	\$7F	Drive number
128	\$80	Track number
129	\$81	Sector number
130	\$82	Channel number
131	\$83	Secondary address
132	\$84	Secondary address
133	\$85	Data byte
139	\$8B-\$8D	Work storage for division
148	\$94-\$95	Actual buffer pointer
153	\$99-\$9A	Address of buffer 0 \$300
155	\$9B-\$9C	Address of buffer 1 \$400
157	\$9D-\$9E	Address of buffer 2 \$500
159	\$9F-\$A0	Address of buffer 4 \$600
161	\$A1-\$A2	Address of buffer 5 \$700
163	\$A3-\$A4	Pointer to input buffer \$200
165	\$A5-\$A6	Pointer to buffer for error message \$2D5

181	\$B5-\$BA	Record # lo, block # lo
187	\$BB-\$C0	Record # hi, block # hi
193	\$C1-\$C6	Write pointer for rel. file
199	\$C7-\$CC	Record length for rel. files
212	\$D4	Pointer in record for rel. file
213	\$D5	Side sector number
214	\$D6	Pointer to data block in side sector
215	\$D7	Pointer to record in rel. file
231	\$E7	File type
249	\$F9	Buffer number
256-325	\$100-\$145	Stack
512-552	\$200-\$228	Buffer for command string
586	\$24A	File type
600	\$258	Record length
601	\$259	Track side-sector
602	\$25A	Sector side-sector
628	\$274	Length of input line
632	\$278	Number of file names
663	\$297	File control method
640-644	\$280-\$284	Track of a file
645-649	\$285-\$289	Sector of a file
725-761	\$2D5-\$2F9	Buffer for error message
762/764	\$2FA/\$2FC	Number of free blocks
768-1023	\$300-\$3FF	Buffer 0
1024-1279	\$400-\$4FF	Buffer 1
1280-1535	\$500-\$5FF	Buffer 2
1536-1791	\$600-\$6FF	Buffer 3
1792-2047	\$700-\$7FF	Buffer 4



## Anatomy of the 1541 Disk Drive

### 3.2 Operation of the DOS - An Overview

The VIC-1541 is an intelligent disk drive with its own microprocessor and control system (Disk Operation System, DOS). This means that no memory space or processing time is taken from the computer. The computer needs only transmit commands to the disk drive, which it then executes on its own.

The disk performs three tasks simultaneously: Firstly, it manages data traffic to and from the computer. Secondly, it interprets the commands and performs the management of files and the associated communications channels and block buffer. Thirdly, it handles the hardware-oriented related functions of the disk drive - formatting, reading and writing, etc.

These tasks are carried out simultaneously by the 6502 microprocessor in the VIC 1541. This is possible with the help of the interrupt technique. Only in this way can three tasks be executed simultaneously.

Most of the DOS is concerned with interpreting and executing the transmitted commands. The reception of data and commands from the computer is controlled by interrupts. If the computer wants to talk to a peripheral device, it sends a pulse along the ATN line (ATtention, see section 5.1). This generates an interrupt at the disk drive. The DOS stops its current task and notices that the computer wants to send data. The DOS then finishes the original task. After that, the DOS will accept further data and commands from the computer. If the command is finished, the DOS stays in a wait loop until new commands arrive from the disk.

The execution of a command at this level is limited to the logical processing of the command, the management of the communications channel to and from the computer and the preparation and retrieval of data to be written or read, respectively. The tasks of a disk controller, formatting diskettes and writing and reading individual blocks, must also be performed by the processor.

These tasks are again interrupt controlled. Regular programs in the disk are interrupted every 14 milliseconds by a built-in timer, and control branches to a program that fulfills the tasks of a disk controller. Communications between the two independent programs is handled through a common area of memory, in which the main program places codes for the disk controller program. If the interrupt program is active, it looks at the memory locations to determine which activities are demanded, such as formatting a diskette. If this is the case, the drive and head motors are set in motion. At the end of the interrupt routine, the main program examines the memory locations to determine if the task was carried out by the disk controller, or if it

must wait yet. In this way, the main program is informed in case of an error, such as a read error or if a write protect tab is present. The main program can then react appropriately and display the error message, for example.

In the large CBM disks, two 6504 microprocessors are used as a disk controller. Communication again occurs over a common area of memory.

An overview of the storage layout of the DOS such as the I/O primitives for managing the diskette and serial bus can be found in the previous section.

This overview of the work of the DOS is naturally just a rough outline. If you want more exact information, refer to the DOS listing of the VIC 1541 in section 3.5, in which the complete 16K control system is documented.

## Anatomy of the 1541 Disk Drive

### 3.3 The Structure of the VIC 1541 Diskette

The diskette of the 1541 is divided into 35 tracks. Each track contains from 17 to 21 sectors. The total number of sectors is 683. Because the directory occupies track 18, 664 data are available for use, each containing 256 bytes. The tracks are laid out as follows:

-----		
: TRACK :	NUMBER OF SECTORS :	
:-----:		
: 1 TO 17 :	21	:
:18 TO 24 :	19	:
:25 TO 30 :	18	:
:31 TO 35 :	17	:
-----		

The varying number of sectors per track is necessitated by the shortening of the tracks from the midpoint on.

#### 3.3.1 The BAM of the VIC 1541

BAM is an abbreviation for Block Availability Map. The BAM indicates whether a block on the diskette is free or allocated to a file. After every manipulation of blocks (saving, deleting, etc.) the BAM is updated. When the BAM indicates that a file to be saved requires more blocks than are available, an error message is given. When a file is OEPNed, the BAM in the DOS storage is updated, and is rewritten to disk when the file is CLOSED. Commands that have a write or delete function read the BAM, update it, and rewrite it to the diskette. The BAM is organized as follows on track 18 sector 0:

-----			
: Track 18, sector 0			:
:-----:			
: BYTE	:	CONTENTS	: MEANING
:-----:			
: 0,1 (\$00-\$01)	:	\$12,\$01	: Track and sector of the 1st :
:	:	:	: block of the directory :
: 2 (\$02)	:	\$41	: ASCII character 'A'; :
:	:	:	: indicates 1541 format :
: 3 (\$03)	:	\$00	: Zero flag for future use :
: 4-143 (\$04-\$8F)	:	:	: Bit map of free and :
:	:	:	: allocated blocks *
:-----:			
: * 1 = block free; 0 = block allocated			:
:-----:			

The bit map of the blocks is organized so that 4 bytes

represent the sectors on a track. As can be inferred from the following table, the first of the 4 bytes contain the number of free blocks in the track. The other 3 bytes (24 bits) indicate which blocks are free and which are allocated in this track.

Structure of the BAM entry of a track:

```

-----
: BYTE      : CONTENTS                                     :
-----
: 0          : Number of available blocks in this track    :
: 1          : Bit map of sectors 0-7                      :
: 2          : Bit map of sectors 8-15                     :
: 3          : Bit map of sectors 16-23                    :
-----

```

4 bytes of a track designation in the BAM:

```

-----
: Track 18, sector 0, bytes 4-7 (track 1) :
-----
: 00001010 : 00000000 00000011 11111111 :
: ($0A)    : ($00)    ($03)    ($FF)    :
-----
: 10 free   : 1 = free                               :
: blocks    : 0 = allocated                           :
-----

```

Using a simple program, you can read the first byte of each track entry in the bit map, add them up and find the total number of free blocks on the diskette.

### 3.3.2 The Directory

The directory is the table of contents of the diskette. It contains the following information:

- disk name
- disk ID
- DOS version number
- filenames
- file types
- blocks per file
- free blocks

This directory is loaded into memory with the command **LOAD "\$",8**. A program previously in memory will be destroyed! It can be displayed on the screen with the **LIST** command.

The directory occupies all of track 18 on the disk. The file entries follow the directory header. Each block accommodates

## Anatomy of the 1541 Disk Drive

a maximum of 8 file entries. Because the BAM and the header occupy one block, 18 blocks are left for file entries. A total of 144 files may reside on one diskette (18 blocks with 8 entries each).

### Format of the directory header:

-----			
: Track 18, sector 0	:	:	:
-----			
: BYTE	: CONTENTS	: MEANING	:
-----			
: 144-161 (\$90-\$A1)	:	: Disk name (padded with	:
:	:	: shifted spaces)	:
: 162,163 (\$A2-\$A3)	:	: Disk ID marker	:
: 164 (\$A4)	: \$A0	: Shifted Space	:
: 165,166 (\$A5-\$A6)	: \$32,\$41	: ASCII characters "2A"	:
:	:	: (format)	:
: 167-170 (\$A7-\$AA)	: \$A0	: Shifted Space	:
: 171-255 (\$AB-\$FF)	: \$00	: not used, filled with 0	:
-----			
: * Bytes 180 to 191 have the contents "BLOCKS FREE" on	:	:	:
: many diskettes	:	:	:
-----			

### The Diskette Name:

The name of the diskette can be a maximum of 16 characters in length and is established when the diskette is formatted. If fewer than 16 characters are given, the rest is filled with shifted spaces (\$A0). The following BASIC routine reads the name and saves it in the string variable DNS:

```
100 OPEN 15,8,15,"I0"           : REM COMMAND CHANNEL 15
                                : AND DISK INITIALIZED
110 OPEN 2,8,2,"#"              : REM DATA CHANNEL 2 OPENED
120 PRINT#15,"B-R";2;0;18;0     : REM TRACK 18, SECTOR 0 READ
                                : AND PLACED IN CHANNEL 2
130 PRINT#15,"B-P";2;144        : REM BUFFER-POINTER TO BYTE
                                : 144
140 DNS=""                      : REM STRING DNS IS ERASED
150 REM LOOP TO READ THE 16 BYTES OF THE NAME
160 FOR I=1 TO 16
170 ::GET#2,X$                  : REM READ A BYTE
180 ::IF ASC(X$)=160 THEN 200    : REM IGNORE SHIFT SPACE
190 ::DNS=DNS+X$                : REM BYTE ADDED TO DNS
200 NEXT I
210 CLOSE 2:CLOSE 15           : REM CLOSE CHANNELS
```

After running the routine, the string DNS contains the disk name.

**Diskette ID:**

The diskette ID is two characters in length and is specified when formatting the diskette. The DOS uses this ID to detect if a diskette in the drive has been replaced. If so, then the DOS performs an INITIALIZE. Initializing a diskette loads the BAM into memory in the drive. This way, the actual BAM is always in memory, provided the ID given when formatting is always different. Should this not be the case, a diskette must be initialized explicitly by using the INITIALIZE command.

**3.3.3 The Directory Format**

Blocks 1 through 19 on track 18 contain the file entries. The first two bytes of a block point to the next directory block with file entries. If no more directory blocks follow, these bytes contain \$00 and \$FF, respectively.

-----			
:	Track 18, sector 1	:	:
-----			
:	Byte	:	Contents
-----			
:	0,1	(\$00,\$01)	: Track and sector number of the
:	:	:	: next directory block
:	2-31	(\$02-\$1F)	: Entry of 1st file
:	34-63	(\$22-\$3F)	: Entry of 2nd file
:	66-95	(\$42-\$5F)	: Entry of 3rd file
:	98-127	(\$62-\$7F)	: Entry of 4th file
:	130-159	(\$82-\$9F)	: Entry of 5th file
:	162-191	(\$A2-\$BF)	: Entry of 6th file
:	194-223	(\$C2-\$DF)	: Entry of 7th file
:	226-255	(\$E2-\$FF)	: Entry of 8th file
-----			

**Format of a Directory Entry:**

Each file entry consists of 30 bytes, the functions of which are described below:

## Anatomy of the 1541 Disk Drive

: BYTE	: CONTENTS	:
: 0 (\$00)	: File type	:
: 1,2 (\$01,\$02)	: Track and sector number of the	:
:	: first data block	:
: 3-18 (\$03-\$12)	: Filename (padded with "SHIFT SPACE"	:
: 19,20 (\$13,\$14)	: Only used for relative files	:
:	: (track and sector of the first	:
:	: side-sector block)	:
: 21 (\$15)	: Only used for relative files	:
:	: (record length)	:
: 22-25 (\$16-\$19)	: Not used	:
: 26,27 (\$1A-\$1B)	: Track and sector number of the new	:
:	: file when overwritten with the @:	:
: 28,29 (\$1C-\$1D)	: Number of blocks in the file (low	:
:	: byte, high byte)	:

### File Type Marker:

Byte 0 of the file entry denotes the file type. Bits 0-2 are used to indicate the 5 file types. Bit 7 indicates if the file has been CLOSED properly. Closing a file sets bit 7. An unclosed file is denoted with an asterisk in front of the file type in the directory listing. If, for example, a sequential file "TEST" is opened and the directory is listed, this file will be represented like this:

```
12      "TEST"          *SEQ
```

If the file is CLOSED again, the asterisk does not appear in future directory listings. If this file remains unclosed and later opened, the error message "WRITE FILE OPEN" will appear.

### The File Type:

In order to understand the function of byte 0 in the file entry, the file type, a table of all file types follows:

: File type	: Bit mask opened	: Bit mask closed	:
:	: 7654 3210    HEX	: 7654 3210    HEX	:
: DELETED	: 0000 0000    \$00	: 1000 0000    \$80	:
: SEQUENTIAL	: 0000 0001    \$01	: 1000 0001    \$81	:
: PROGRAM	: 0000 0010    \$02	: 1000 0010    \$82	:
: USER	: 0000 0011    \$03	: 1000 0011    \$83	:
: RELATIVE	: 0000 0100    \$04	: 1000 0100    \$84	:

Perhaps you have noticed that bits 3-6 have no function. But we verified with help from the DOS listing, bit 6 has a

function:

#### BIT 6 OF THE FILE TYPE DENOTES A PROTECTED FILE!

If you set this bit to 1, the corresponding file can no longer be deleted. This is designated in the directory listing with a < next to the file type. Because setting this bit requires some complicated commands, you will find a program in chapter 4 of this book with which you can protect, unprotect, and delete files.

#### Track and sector of the first Data Block

Bytes 1 and 2 of the file entry point to the first data block of the file. The first byte contains the track and the second the sector number where the file begins. The first data block, in turn contains a pointer to the second block of the file (also contained in the first two bytes of the block). The last data block of the file is indicated by a first-byte value of \$00. The second byte contains the number of bytes used in this last sector.

This concatenation can be explained with the help of the DOS MONITOR, contained in this book:

```
>:B0 A0 A0 A0 A0 A0 00 00 00 ...
>:B8 00 00 00 00 00 00 0B 00 .....
>:C0 00 00 81 13 09 54 31 32 .....T12
>:C8 2F 53 30 31 A0 A0 A0 A0 /S01
>:D0 A0 A0 A0 A0 A0 00 00 00 ...
>:D8 00 00 00 00 00 00 06 00 .....
>:E0 00 00 82 10 00 44 49 53 .....DIS
>:E8 4B 20 41 44 44 52 20 43 K ADDR C
>:F0 48 41 4E 47 45 00 00 00 HANGE...
>:F8 00 00 00 00 00 00 04 00
```

This is an extract from the directory (track 18, sector 1) of the TEST/DEMO diskette. You can follow the organization of the file DISK ADDR CHANGE. The entry of this file begins at byte \$E2 and ends with byte \$FF. This is a PRG file, which can be recognized by the file type \$82 in byte \$E2. This file comprises 4 blocks on the disk. This is evident from bytes \$FE and \$FF. Bytes \$E3 and \$E4 of the entry address the first data block of the file (\$10, \$00, corresponding to track 16, sector 0).

Let's look at a section of this block:

```
>:00 10 0A 01 04 0F 04 64 00 .....$.
>:08 97 35 39 34 36 38 2C 31 .59468,1
>:10 32 00 39 04 6E 0D 99 22 2.9...."
>:18 93 13 11 11 11 11 44 52 .....DR
>:20 49 56 45 20 41 44 44 52 IVE ADDR
>:28 45 53 53 20 43 48 41 4E ESS CHAN
```



## Anatomy of the 1541 Disk Drive

```
>:30 47 45 20 50 52 4F 47 52 GE PROGR
>:38 41 4D 22 00 59 04 6F 00 AM".Y./
>:40 99 22 11 54 55 52 4E 20 ".TURN
>:48 4F 46 46 20 41 4C 4C 20 OFF ALL
```

This block contains the first part of the program. It is stored on the diskette exactly as it is stored in the computer's memory. The BASIC commands are converted to one byte codes called tokens. This is why only the text can be recognized in the right hand translation of the hexadecimal codes. The first two bytes of this data block indicate the second data block (\$10 and \$0A, track 16, sector 10) from which this section follows:

```
>:00 10 14 34 30 00 1D 05 A0 ..40...
>:08 00 8D 20 33 30 30 3A 20 .. 300:
>:10 8F 20 46 49 4E 44 20 44 . FIND D
>:18 52 49 56 45 20 54 59 50 DRIVE TYP
>:20 45 00 39 05 AA 00 8D 20 E.9. ..
>:28 36 30 30 3A 20 8F 20 43 600: . C
>:30 48 41 4E 47 45 20 41 44 HANGE AD
>:38 44 52 45 53 53 00 68 05 DRESS.(
>:40 B4 00 99 22 11 54 48 45 ..".THE
>:48 20 53 45 4C 45 43 54 45 SELECTE
```

The program is continued in this block. Bytes \$00 and \$01 point to the third data block of the file (\$10, \$14, track 16, sector 20):

```
>:00 10 08 31 30 30 30 00 23 ..1000.#
>:08 06 54 01 8B 20 43 B2 32 .T.. C 2
>:10 35 34 20 A7 20 4D 54 B2 54 MT
>:18 31 31 39 3A 20 8F 3A 20 119: ..
>:20 32 30 33 31 20 56 32 2E 2031 V2.
>:28 36 00 45 06 5E 01 8B 20 6.E. ..
>:30 43 B2 32 32 36 20 A7 20 C 226
>:38 4D 54 B2 35 30 3A 20 8F MT 50: .
>:40 3A 20 32 30 34 30 20 56 : 2040 V
>:48 31 2E 32 00 67 06 68 01 1.2. .(.
```

This is the next to the last block of the program. You have no doubt recognized that the data blocks are in the same track, but are not contiguously. The first data block is block 0. The next is block 10, 10 blocks from the first block. 9 blocks are always skipped between data blocks of a file. The third data block is block number 20. The DOS begins again with the first block if the calculated block oversteps the highest block. Because track 16 contains 21 blocks, the last data block is block number 8. The first two bytes of this third block address it:

```
>:00 00 F8 5A 42 B2 31 20 A7 . ZB 1
>:08 20 34 34 30 00 14 07 A3 440...
>:10 01 8B 20 53 54 20 A7 20 .. ST
>:18 31 30 30 30 00 45 07 B8 1000.E.
```

```
>:20 01 98 31 35 2C 22 4D 2D ..15,"M-
>:28 52 22 C7 28 31 37 32 29 R" (172)
>:30 C7 28 31 36 29 3A A1 23 (16): #
>:38 31 35 2C 5A 43 24 3A 5A 15,ZC$:Z
>:40 43 B2 C6 28 5A 43 24 AA C F(2C$:
>:48 C7 28 30 29 29 00 66 07 G(0)).&.
```

Here the end of the program is marked by the value \$00 in byte \$00. Byte \$01 gives the number of bytes in this last block that belong to the program. (\$F8 corresponds to 248 bytes). Now we can find out the size of the program:

```
3 blocks with 254 bytes each = 762 bytes
last block                    = 248 bytes
-----
Size of the program           1100 bytes
=====
```

## The Filename:

The filename is contained in bytes 3-18 of the file entry. It consists of a maximum of 16 characters. Should the name be shorter than 16 characters, the rest of the name is padded with shifted spaces (\$A0).

## Track and Sector of the new File for "Overwriting":

If a file is overwritten by using the @:, the new file is first completely saved. No filename entry is made in the directory for this file because the file already exists under this same name. Instead the address of the first block of the new file is placed in bytes 26 and 27 of the filename entry. If the new program is removed, the old one is deleted, which merely designates the blocks allocated to the file as free in the BAM. Now the address of the first data block of the new file is placed into the filename entry in bytes 1 and 2 is used and the file is "overwritten".

## Number of Blocks in the File:

The length of a file is given in bytes 28 and 29 of its file entry. A file consists of at least one block and as many as 664 blocks. The first byte is the low byte, and the second is the high byte. If, for example, you discovered the file length \$1F,\$00 with the DISK MONITOR, the file consists of 31 blocks.

## Anatomy of the 1541 Disk Drive

### 3.4 The Organization of Relative Files

Relative files differ from sequential files in that each data record can be accessed directly by a record number. The 1541 DOS takes care of most of the tasks required to support relative records. Let's take a closer look at the organization of a relative file.

First OPEN a relative file with a record length of 100:

```
OPEN 2,8,2, "REL-FILE,L,"+CHR$(100)
```

Now write data record number 70:

```
OPEN 1,8,15
PRINT#1,"P"+CHR$(2)+CHR$(70)+CHR$(0)+CHR$(1)
PRINT#2,"DATA FOR RECORD 70"
CLOSE 2 : CLOSE 1
```

The directory entry then looks like this:

```
>:00 .. .. 84 11 00 52 45 4C ...REL
>:08 2D 46 49 4C 45 A0 A0 A0 -FILE
>:10 A0 A0 A0 A0 A0 11 0A 64 ...$
>:18 00 00 00 00 00 00 1D 00 .....
```

The first byte \$84 denotes a relative file. The next two bytes denote the first track and sector of the data (\$11, \$00; track 17 sector 0); exactly as with a sequential file. As usual, the name of the file follows (16 characters, padded with shifted spaces, \$A0). Following are two fields not used with sequential files. The first field is a two byte pointer to the track and sector of the first **side-sector** block. A side-sector contains the pointers to each data record and is described more in detail later (\$11, \$0A; track 17, sector 10). The second field is a byte which contains the record length, a value between 1 and 254, in our case \$64 (100).

The convenience of being able to access each record individually requires a definite length for each record that must be defined when establishing a relative file. The rest of the fields in the directory entry have the usual significance; the last two bytes contain the number of blocks in the file (lo and hi byte, \$1D and \$00 (29)).

What does such a side-sector block look like and what is its function?

The side-sector blocks contain the track and sector pointers to the individual data records. For example, if we want to read the 70th record in the relative file, the DOS consults the side-sector block to determine which track and sector contains the record and then read this record directly. As

a result, you can read the 70th record of the file without having to read the entire file. Now let's take a look at the exact construction of a side-sector block. This side-sector block is from our previous file.

```
>:00 00 47 00 64 11 0A 00 00 .G.$....
>:08 00 00 00 00 00 00 00 00 .....
>:10 11 00 11 0B 11 01 11 0C .....
>:18 11 02 11 0D 11 03 11 0E .....
>:20 11 04 11 0F 11 05 11 10 .....
>:28 11 06 11 11 11 07 11 12 .....
>:30 11 08 11 13 11 09 11 14 .....
>:38 10 08 10 12 10 06 10 10 .....
>:40 10 04 10 0E 10 02 10 0C .....
>:48 00 00 00 00 00 00 00 00 .....
>:50 00 00 00 00 00 00 00 00 .....
etc.
```

The first two bytes point to the track and sector of the next side-sector block, as usual. In our case, no further side-sector blocks exist (\$00) and only \$47 = 71 bytes of this sector are used. Byte 2 contains the number of the side-sector block, 00. A relative file can contain a maximum of 6 such blocks; the numbering goes from 0 to 5. The record length, \$64 (100), is in byte 3. The next twelve bytes (bytes 4 through 15) contain the track and sector pointers (two bytes each) to the 6 side-sector blocks (00,00 means the block is not yet used). Starting at byte 16 (\$10) are the pointers to the data, and the track and sector pointers to the first 120 data blocks (in our case, only 28 pointers). Using the record number and record length, the DOS can calculate in which block the data lies and at which position within the block the record begins. Take the following example, for instance:

To read the 70th record from the file with a record length of 100 characters, you can perform the following calculations:

$$(70-1) * 100 / 254$$

We get a quotient of 27 and a remainder of 42. The DOS now knows that the record can be found in the 27th data block at the 42+2 or 44th position.

Here's an explanation of the calculation. Each block contains 256 bytes, the first two of which are used as a pointer to the next block. 254 bytes are then left over for data storage. We can calculate the byte number from the start of the file (which is record 1) from the record number and record length. If we divide this value by the number of bytes per block, we get the number of the block containing the record. The remainder of the division gives the position within the block (add 2, because the first two bytes serve as a pointer). If the record overlaps the end of the block,

## Anatomy of the 1541 Disk Drive

the next block must also be read.

In our example, the 27th data block lies in track \$10 = 16 and sector \$0C = 12. If we read this block, we get the following picture:

```
>:00 00 F3 00 00 00 00 00 00 .....
>:08 00 00 00 00 00 00 00 00 .....
>:10 00 00 00 00 00 00 00 00 .....
>:18 00 00 00 00 00 00 00 00 .....
>:20 00 00 00 00 00 00 00 00 .....
>:28 00 00 00 00 44 41 54 41 ....DATA
>:30 20 46 4E 52 20 52 45 43   FOR REC
>:38 46 52 44 20 37 30 0D 00  ORD 70..
>:40 00 00 00 00 00 00 00 00 .....
>:48 00 00 00 00 00 00 00 00 .....
>:50 00 00 00 00 00 00 00 00 .....
>:58 00 00 00 00 00 00 00 00 .....
>:60 00 00 00 00 00 00 00 00 .....
>:68 00 00 00 00 00 00 00 00 .....
>:70 00 00 00 00 00 00 00 00 .....
>:78 00 00 00 00 00 00 00 00 .....
>:80 00 00 00 00 00 00 00 00 .....
>:88 00 00 00 00 00 00 00 00 .....
>:90 FF 00 00 00 00 00 00 00 .....
>:98 00 00 00 00 00 00 00 00 .....
>:A0 00 00 00 00 00 00 00 00 .....
>:A8 00 00 00 00 00 00 00 00 .....
>:B0 00 00 00 00 00 00 00 00 .....
>:B8 00 00 00 00 00 00 00 00 .....
>:C0 00 00 00 00 00 00 00 00 .....
>:C8 00 00 00 00 00 00 00 00 .....
>:D0 00 00 00 00 00 00 00 00 .....
>:D8 00 00 00 00 00 00 00 00 .....
>:E0 00 00 00 00 00 00 00 00 .....
>:E8 00 00 00 00 00 00 00 00 .....
>:F0 00 00 00 00 FF 00 00 00 .....
>:F8 00 00 00 00 00 00 00 00 .....
```

If we get a block number greater than 120 from the calculation, the pointer can no longer be found on the first side-sector block, rather in the next side-sector blocks. In this case, you divide the block number by 120, the quotient being the number of the side-sector block. The remainder gives the location of the pointer within this block. For instance, to find record number 425, divide by 120 and get a quotient 3, remainder 65. Therefore, you must read side-sector block 3 and get the pointer to the 65th data block. Between 2 and 4 block accesses are necessary to access a record of a relative data file.

When creating or expanding a relative file, the following takes place:

First, a directory entry is created for the relative file,

containing the record length. Two channels are reserved for the relative file, one for the data, the other for the side-sectors. If a record pointer is set to a specific record, the DOS first checks to see if the record already exists. If so, the corresponding block is read and the buffer pointer set so that the contents can be accessed. If not, the record is created. All records preceding this record number that do not already exist are also created. The first byte of a new record is written to contain \$FF (255), and the rest of the record is filled with \$00.

If the corresponding record is at the beginning of a block, the rest of the block is filled with empty records. Each time a non-existing record is accessed, the error message **50, RECORD NOT PRESENT** is returned. When writing a new record, this is not considered an error, but indicates that a new record was created.

You can use this method for creating a new file if you know the maximum number of data records. You simply set the record pointer to this record and write \$FF (CHR\$(255)) to this record. By allocating a file like this, the error message 50 no longer appears. You also know if there is sufficient space on the diskette. If not, the error message **52, FILE TOO LARGE** is returned.

With a maximum of 6 side sectors, a relative file can contain  $6 * 120 * 254 = 182,880$  bytes. In the case of the VIC 1541, this is more than the capacity of the whole diskette. With the bigger 8050 drive, which contains more than 500K of storage, this may present a limitation. But DOS version 2.7 has an expansion of the side-sector procedure ('super side-sector'), with which a relative file may contain up to 23 MB. DOS 2.7 is contained in the CBM 8250 and the Commodore hard drives as well as the newer 8050 drives (see section 5.2).

Because a relative file requires two data channels, and the VIC 1541 has only 3 channels available, only one relative file can be open at a time. The third channel can still be used for a sequential file open at the same time. With the larger CBM drives, more channels are available (3 relative files open simultaneously, see also section 5.2).

## Anatomy of the 1541 Disk Drive

### 3.5 DOS 2.6 ROM LISTINGS

```
***** turn LED on
C100  78          SEI
C101  A9 F7      LDA #$F7      erase LED bit
C103  2D 00 1C   AND $1C00
C106  48          PHA
C107  A5 7F      LDA $7F      drive number
C109  F0 05      BEQ $C110     0?
C10B  68          PLA
C10C  09 00      ORA #$00      not drive 0, turn LED off
C10E  D0 03      BNE $C113
C110  68          PLA
C111  09 08      ORA #$08      turn LED on
C113  8D 00 1C   STA $1C00
C116  58          CLI
C117  60          RTS

***** turn LED on
C118  78          SEI
C119  A9 08      LDA #$08
C11B  0D 00 1C   QRA $1C00     LED on
C11E  8D 00 1C   STA $1C00
C121  58          CLI
C122  60          RTS

***** erase error flags.
C123  A9 00      LDA #$00
C125  8D 6C 02   STA $026C
C128  8D 6D 02   STA $026D
C12B  60          RTS

*****
C12C  78          SEI
C12D  8A          TXA          save X register
C12E  48          PHA
C12F  A9 50      LDA #$50
C131  8D 6C 02   STA $026C
C134  A2 00      LDX #$00
C136  BD CA FE   LDA $FECA,X   8
C139  8D 6D 02   STA $026D
C13C  0D 00 1C   ORA $1C00
C13F  8D 00 1C   STA $1C00     turn LED on
C142  68          PLA
C143  AA          TAX          get x register back
C144  58          CLI
C145  60          RTS

***** interpret command from
computer
C146  A9 00      LDA #$00
C148  8D F9 02   STA $02F9
C14B  AD 8E 02   LDA $028E     last drive number
```

# Anatomy of the 1541 Disk Drive

C14E	85	7F		STA \$7F	drive number
C150	20	BC	E6	JSR \$E6BC	prepare 'ok' message
C153	A5	84		LDA \$84	secondary address
C155	10	09		BPL \$C160	
C157	29	0F		AND #\$0F	
C159	C9	0F		CMP #\$0F	15, command channel
C15B	F0	03		BEQ \$C160	yes
C15D	4C	B4	D7	JMP \$D7B4	to OPEN command
C160	20	B3	C2	JSR \$C2B3	determine line length and erase flags
C163	B1	A3		LDA (\$A3),Y	get first character
C165	8D	75	02	STA \$0275	and store
C168	A2	0B		LDX #\$0B	11
C16A	BD	89	FE	LDA \$FE89,X	commands
C16D	CD	75	02	CMP \$0275	compare to first character
C170	F0	08		BEQ \$C17A	found?
C172	CA			DEX	
C173	10	F5		BPL \$C16A	
C175	A9	31		LDA #\$31	not found
C177	4C	C8	C1	JMP \$C1C8	31, 'syntax error'
C17A	8E	2A	02	STX \$022A	number of command words
C17D	E0	09		CPX #\$09	
C17F	90	03		BCC \$C184	command number < 9?
C181	20	EE	C1	JSR \$C1EE	test for 'R', 'S', and 'N'
C184	AE	2A	02	LDX \$022A	command number
C187	BD	95	FE	LDA \$FE95,X	jump address 10
C18A	85	6F		STA \$6F	
C18C	BD	A1	FE	LDA \$FEA1,X	jump address h1
C18F	85	70		STA \$70	
C191	6C	6F	00	JMP (\$006F)	jump to command
*****					prepare error message after executing command
C194	A9	00		LDA #\$00	
C196	8D	F9	02	STA \$02F9	
C199	AD	6C	02	LDA \$026C	flag set?
C19C	D0	2A		BNE \$C1C8	yes, then set error message
C19E	A0	00		LDY #\$00	
C1A0	98			TYA	error number 0
C1A1	84	80		STY \$80	track number 0
C1A3	84	81		STY \$81	sector number 0
C1A5	84	A3		STY \$A3	
C1A7	20	C7	E6	JSR \$E6C7	prepare 'ok' message
C1AA	20	23	C1	JSR \$C123	erase error flag
C1AD	A5	7F		LDA \$7F	drive number
C1AF	8D	8E	02	STA \$028E	save as last drive number
C1B2	AA			TAX	
C1B3	A9	00		LDA #\$00	
C1B5	95	FF		STA \$FF,X	
C1B7	20	BD	C1	JSR \$C1BD	erase input buffer
C1BA	4C	DA	D4	JMO \$D4DA	close internal channel
*****					erase input buffer
C1BD	A0	28		LDY #\$28	erase 41 characters
C1BF	A9	00		LDA #\$00	



# Anatomy of the 1541 Disk Drive

C1C1	99 00 02	STA \$0200,Y	\$200 to \$228
C1C4	88	DEY	
C1C5	10 FA	BPL \$C1C1	
C1C7	60	RTS	
*****			give error message (track & sector)
C1C8	A0 00	LDY #\$00	
C1CA	84 80	STY \$80	track = 0
C1CC	84 81	STY \$81	sector = 0
C1CE	4C 45 E6	JMP \$E645	error number acc, generate error message
*****			check input line
C1D1	A2 00	LDX #\$00	
C1D3	8E 7A 02	STX \$027A	pointer to drive number
C1D6	A9 3A	LDA #\$3A	':'
C1D8	20 68 C2	JSR \$C268	test line to ':' or to end
C1DB	F0 05	BEQ \$C1E2	no colon found?
C1DD	88	DEY	
C1DE	88	DEY	
C1DF	8C 7A 02	STY \$027A	point to drive number (before colon)
C1E2	4C 68 C3	JMP \$C368	get drive # and turn LED on
*****			check input line
C1E5	A0 00	LDY #\$00	pointer to input buffer
C1E7	A2 00	LDX #\$00	counter for commas
C1E9	A9 3A	LDA #\$3A	':'
C1EB	4C 68 C2	JMP \$C268	test line to colon or to end
*****			check input line
C1EE	20 E5 C1	JSR \$C1E5	test line to ':' or end
C1F1	D0 05	BNE \$C1F8	colon found?
C1F3	A9 34	LDA #\$34	
C1F5	4C C8 C1	JMP \$C1C8	34, 'syntax error'
C1F8	88	DEY	
C1F9	88	DEY	
C1FA	8C 7A 02	STY \$027A	set pointer to colon
C1FD	8A	TXA	position of the drive no.
C1FE	D0 F3	BNE \$C1F3	comma before the colon
C200	A9 3D	LDA #\$3D	yes, then 'syntax error'
C202	20 68 C2	JSR \$C268	'='
C205	8A	TXA	check input to '='
C206	F0 02	BEQ \$C20A	comma found?
C208	A9 40	LDA #\$40	no
C20A	09 21	ORA #\$21	bit 6
C20C	8D 8B 02	STA \$028B	and set bit 0 and 5
C20F	E8	INX	flag for syntax check
C210	8E 77 02	STX \$0277	
C213	8E 78 02	STX \$0278	
C216	AD 8A 02	LDA \$028A	wildcard found?
C219	F0 0D	BEQ \$C228	no
C21B	A9 80	LDA #\$80	
C21D	0D 8B 02	ORA \$028B	set bit 7
C220	8D 8B 02	STA \$028B	

# Anatomy of the 1541 Disk Drive

C223	A9 00	LDA #\$00	
C225	8D 8A 02	STA \$028A	reset wildcard flag
C228	98	TYA	'=' found?
C229	F0 29	BEQ \$C254	no
C22B	9D 7A 02	STA \$027A,X	
C22E	AD 77 02	LDA \$0277	number of commas before '='
C231	8D 79 02	STA \$0279	
C234	A9 8D	LDA #\$8D	shift CR
C236	20 68 C2	JSR \$C268	check line to end
C239	E8	INX	increment comma counter
C23A	8E 78 02	STX \$0278	store # of commas
C23D	CA	DEX	
C23E	AD 8A 02	LDA \$028A	wildcard found?
C24A	F0 02	BEQ \$C245	no
C243	A9 08	LDA #\$08	set bit 3
C245	EC 77 02	CPX \$0277	comma after '='?
C248	F0 02	BEQ \$C24C	no
C24A	09 04	ORA #\$04	set bit 2
C24C	09 03	ORA #\$03	set bits 0 and 1
C24E	4D 8B 02	EOR \$028B	
C251	8D 8B 02	STA \$028B	as flag for syntax check
C254	AD 8B 02	LDA \$028B	syntax flag
C257	AE 2A 02	LDX \$022A	command number
C25A	3D A5 FE	AND \$FEA5,X	combine with check byte
C25D	D0 01	BNE \$C260	
C25F	60	RTS	
C260	8D 6C 02	STA \$026C	set error flag
C263	A9 30	LDA #\$30	
C265	4C C8 C1	JMP \$C1C8	30, 'syntax error'

*****			search characters in input buffer
C268	8D 75 02	STA \$0275	save character
C26B	CC 74 02	CPY \$0274	already done?
C26E	B0 2E	BCS \$C29E	yes
C270	B1 A3	LDA (\$A3),Y	get char from buffer
C272	C8	INY	
C273	CD 75 02	CMP \$0275	compared with char
C276	F0 28	BEQ \$C2A0	found
C278	C9 2A	CMP #\$2A	'*'
C27A	F0 04	BEQ \$C280	
C27C	C9 3F	CMP #\$3F	'?'
C27E	D0 03	BNE \$C283	
C280	EE 8A 02	INC \$028A	set wildcard flag
C283	C9 2C	CMP #\$2C	','
C285	D0 E4	BNE \$C26B	
C287	98	TYA	
C288	9D 7B 02	STA \$027B,X	note comma position
C28B	AD 8A 02	LDA \$028A	wildcard flag
C28E	29 7F	AND #\$7F	
C290	F0 07	BEQ \$C299	no wildcard
C292	A9 80	LDA #\$80	
C294	95 E7	STA \$E7,X	note flag
C296	8D 8A 02	STA \$028A	and save as wildcard flag
C299	E8	INX	inc comma counter

## Anatomy of the 1541 Disk Drive

C29A	E0 04	CPX #\$04	4 commas already?
C29C	90 CD	BCC \$C26B	no, continue
C29E	A0 00	LDY #\$00	
C2A0	AD 74 02	LDA \$0274	set flag for line end
C2A3	9D 7B 02	STA \$027B,X	
C2A6	AD 8A 02	LDA \$028A	wildcard flag
C2A9	29 7F	AND #\$7F	
C2AB	F0 04	BEO \$C2B1	no wildcard
C2AD	A9 80	LDA #\$80	
C2AF	95 E7	STA \$E7,X	set flag
C2B1	98	TYA	
C2B2	60	RTS	

*****			check line length
C2B3	A4 A3	LDY \$A3	ptr to command input buffer
C2B5	F0 14	BEO \$C2CB	zero?
C2B7	88	DEY	
C2B8	F0 10	BEO \$C2CA	one?
C2BA	B9 00 02	LDA \$0200,Y	pointer to input buffer
C2BD	C9 0D	CMP #\$0D	'CR'
C2BF	F0 0A	BEO \$C2CB	yes, line end
C2C1	88	DEY	
C2C2	B9 00 02	LDA \$0200,Y	preceding character
C2C5	C9 0D	CMP #\$0D	'CR'
C2C7	F0 02	BEO \$C2CB	yes
C2C9	C8	INY	
C2CA	C8	INY	pointer to old value again
C2CB	8C 74 02	STY \$0274	same line length
C2CE	C0 2A	CPY #\$2A	compare with 42 characters
C2D0	A0 FF	LDY #\$FF	
C2D2	90 08	BCC \$C2DC	smaller, ok
C2D4	8C 2A 02	STY \$022A	
C2D7	A9 32	LDA #\$32	
C2D9	4C C8 C1	JMP \$C1C8	32, 'syntax error' line too long

*****			erase flag for input command
C2DC	A0 00	LDY #\$00	
C2DE	98	TYA	
C2DF	85 A3	STA \$A3	pointer to input buffer lo
C2E1	8D 58 02	STA \$0258	record length
C2E4	8D 4A 02	STA \$024A	file type
C2E7	8D 96 02	STA \$0296	
C2EA	85 D3	STA \$D3	
C2EC	8D 79 02	STA \$0279	comma counter
C2EF	8D 77 02	STA \$0277	"
C2F2	8D 78 02	STA \$0278	"
C2F5	8D 8A 02	STA \$028A	wildcard flag
C2F8	8D 6C 02	STA \$026C	error flag
C2FB	A2 05	LDX #\$05	
C2FD	9D 79 02	STA \$0279,X	flags for line analysis
C300	95 D7	STA \$D7,X	directory sectors
C302	95 DC	STA \$DC,X	buffer pointer
C304	95 E1	STA \$E1,X	drive number
C306	95 E6	STA \$E6,X	wildcard flag

# Anatomy of the 1541 Disk Drive

C308	9D 7F 02	STA \$027F,X	track number
C30B	9D 84 02	STA \$0284,X	sector number
C30E	CA	DEX	
C30F	D0 EC	BNE \$C2FD	
C311	60	RTS	
*****			preserve drive number
C312	AD 78 02	LDA \$0278	number of commas
C315	8D 77 02	STA \$0277	save
C318	A9 01	LDA #\$01	
C31A	8D 78 02	STA \$0278	number of drive numbers
C31D	8D 79 02	STA \$0279	
C320	AC 8E 02	LDY \$028E	last drive number
C323	A2 00	LDX #\$00	
C325	86 D3	STX \$D3	
C327	BD 7A 02	LDA \$027A,X	position of the colon
C32A	20 3C C3	JSR \$C33C	get drive no. before colon
C32D	A6 D3	LDX \$D3	
C32F	9D 7A 02	STA \$027A	save exact position
C332	98	TYA	
C333	95 E2	STA \$E2,X	drive number in table
C335	E8	INX	
C336	EC 78 02	CPX \$0278	got all drive numbers?
C339	90 EA	BCC \$C325	no, continue
C33B	60	RTS	
*****			search for drive number
C33C	AA	TAX	note position
C33D	A0 00	LDY #\$00	
C33F	A9 3A	LDA #\$3A	':'
C341	DD 01 02	CMP \$0201,X	colon behind it?
C344	F0 0C	BEQ \$C352	yes
C346	DD 00 02	CMP \$0200,X	colon here?
C349	D0 16	BNE \$C361	no
C34B	E8	INX	
C34C	98	TYA	
C34D	29 01	AND #\$01	drive number
C34F	A8	TAY	
C350	8A	TXA	
C351	60	RTS	
C352	BD 00 02	LDA \$0200,X	get drive number
C355	E8	INX	
C356	E8	INX	
C357	C9 30	CMP #\$30	'0'?
C359	F0 F2	BEQ \$C34D	yes
C35B	C9 31	CMP #\$31	'1'?
C35D	F0 EE	BEQ \$C34D	yes
C35F	D0 EB	BNE \$C34C	no, use last drive number
C361	98	TYA	last drive number
C362	09 80	ORA #\$80	set bit 7, uncertain drive #
C364	29 81	AND #\$81	erase remaining bits
C366	D0 E7	BNE \$C34F	
*****			get drive number

# Anatomy of the 1541 Disk Drive

C368	A9 00	LDA #S00	
C36A	8D 8B 02	STA \$028B	erase syntax flag
C36D	AC 7A 02	LDY \$027A	position in command line
C370	B1 A3	LDA (\$A3),Y	get chars from command buffer
C372	20 BD C3	JSR \$C3BD	get drive number
C375	10 11	BPL \$C388	certain number?
C377	C8	INY	increment pointer
C378	CC 74 02	CPY \$0274	line end?
C37B	B0 06	BCS \$C383	yes
C37D	AC 74 02	LDY \$0274	
C380	88	DEY	
C381	D0 ED	BNE \$C370	search line for drive no.
C383	CE 8B 02	DEC \$028B	
C386	A9 00	LDA #S00	
C388	29 01	AND #S01	
C38A	85 7F	STA \$7F	drive number
C38C	4C 00 C1	JMP \$C100	turn LED on
*****			
C38F	A5 7F	LDA \$7F	reverse drive number
C391	49 01	EOR #S01	drive number
C393	29 01	AND #S01	switch bit 0
C395	85 7F	STA \$7F	
C397	60	RTS	
*****			
C398	A0 00	LDY #S00	establish file type
C39A	AD 77 02	LDA \$0277	'=' found?
C39D	CD 78 02	CMP \$0278	
C3A0	F0 16	BEO \$C3B8	no
C3A2	CE 78 02	DEC \$0278	get pointer
C3A5	AC 78 02	LDY \$0278	
C3A8	B9 7A 02	LDA \$027A,Y	set pointer to character behind '='
C3AB	A8	TAY	
C3AC	B1 A3	LDA (\$A3),Y	pointer to buffer
C3AE	A0 04	LDY #S04	compare with marker for file type
C3B0	D9 BB FE	CMP \$FEBB,Y	'S', 'P', 'U', 'R'
C3B3	F0 03	BEO \$C3B8	agreement
C3B5	88	DEY	
C3B6	D0 F8	BNE \$C3B0	
C3B8	98	TYA	
C3B9	8D 96 02	STA \$0296	note file type (1-4)
C3BC	60	RTS	
*****			
C3BD	C9 30	CMP #S30	check drive number
C3BF	F0 06	BEO \$C3C7	'0'
C3C1	C9 31	CMP #S31	'1'
C3C3	F0 02	BEO \$C3C7	
C3C5	09 80	ORA #S80	no zero or one, then set bit 7
C3C7	29 81	AND #S81	
C3C9	60	RTS	

```

***** verify drive number
C3CA  A9 00      LDA #$00
C3CC  85 6F      STA $6F
C3CE  8D 8D 02   STA $028D
C3D1  48         PHA
C3D2  AE 78 02   LDX $0278      number of drive numbers
C3D5  68         PLA
C3D6  05 6F      ORA $6F
C3D8  48         PHA
C3D9  A9 01      LDA #$01
C3DB  85 6F      STA $6F
C3DD  CA         DEX
C3DE  30 0F      BMI $C3EF
C3E0  B5 E2      LDA SE2,X
C3E2  10 04      BPL $C3E8
C3E4  06 6F      ASL $6F
C3E6  06 6F      ASL $6F
C3E8  4A         LSR A
C3E9  90 EA      RCC $C3D5
C3EB  06 6F      ASL $6F
C3ED  D0 E6      BNE $C3D5
C3EF  68         PLA
C3F0  AA         TAX
C3F1  BD 3F C4   LDA $C43F,X    get syntax flag
C3F4  48         PHA
C3F5  29 03      AND #$03
C3F7  8D 8C 02   STA $028C
C3FA  68         PLA
C3FB  0A         ASL A
C3FC  10 3E      BPL $C43C
C3FE  A5 E2      LDA $E2
C400  29 01      AND #$01      isolate drive number
C402  85 7F      STA $7F
C404  AD 8C 02   LDA $028C
C407  F0 28      BEQ $C434
C409  20 3D C6   JSR $C63D      initialize drive
C40C  F0 12      BEQ $C420      error?
C40E  20 8F C3   JSR $C38F      switch to other drive
C411  A9 00      LDA #$00
C413  8D 8C 02   STA $028C
C416  20 3D C6   JSR $C63D      initialize drive
C419  F0 1E      BEQ $C439      no error?
C41B  A9 74      LDA #$74
C41D  20 C8 C1   JSR $C1C8      74, 'drive not ready'
C420  20 8F C3   JSR $C38F

C423  20 3D C6   JSR $C63D      initialize drive
C426  08         PHP
C427  20 8F C3   JSR $C38F      switch to other drive
C42A  28         PLP
C42B  F0 0C      BEQ $C439      no error?
C42D  A9 00      LDA #$00
C42F  8D 8C 02   STA $028C      number of drives
C432  F0 05      BEQ $C439
C434  20 3D C6   JSR $C63D      initialize drive

```

# Anatomy of the 1541 Disk Drive

C437	D0 E2	BNE \$C41B	error?
C439	4C 00 C1	JMP \$C100	Turn LED on
C43C	2A	ROL A	drive # from carry after bit 0
C43D	4C 00 C4	JMP \$C400	
*****			flags for drive check
C440	00 80 41 01 01 01 01 81		
C448	81 81 81 42 42 42 42		
*****			search for file in directory
C44F	20 CA C3	JSR \$C3CA	initialize drive
C452	A9 00	LDA #\$00	
C454	8D 92 02	STA \$0292	pointer
C457	20 AC C5	JSR \$C5AC	read first directory block
C45A	D0 19	BNE \$C475	entry present?
C45C	CE 8C 02	DEC \$028C	drive number clear?
C45F	10 01	BPL \$C462	no
C461	60	RTS	
C462	A9 01	LDA #\$01	
C464	8D 8D 02	STA \$028D	
C467	20 8F C3	JSR \$C38F	change drive
C46A	20 00 C1	JSR \$C100	Turn LED on
C46D	4C 52 C4	JMP \$C452	and search
C470	20 17 C6	JSR \$C617	search next file in directory
C473	F0 10	BEQ \$C485	not found?
C475	20 D8 C4	JSR \$C4D8	verify directory entry
C478	AD 8F 02	LDA \$028F	
C47B	F0 01	BEQ \$C47E	more files?
C47D	60	RTS	
C47E	AD 53 02	LDA \$0253	
C481	30 ED	BMI \$C470	file not found?
C483	10 F0	BPL \$C475	yes
C485	AD 8F 02	LDA \$028F	
C488	F0 D2	BEQ \$C45C	
C48A	60	RTS	
C48B	20 04 C6	JSR \$C604	search next directory block
C48E	F0 1A	BEQ \$C4AA	not found?
C490	D0 28	BNE \$C4BA	
C492	A9 01	LDA #\$01	
C494	8D 8D 02	STA \$028D	
C497	20 8F C3	JSR \$C38F	change drive
C49A	20 00 C1	JSR \$C100	turn LFD on
C49D	A9 00	LDA #\$00	
C49F	8D 92 02	STA \$0292	
C4A2	20 AC C5	JSR \$C5AC	read directory block
C4A5	D0 13	BNE \$C4BA	found?
C4A7	8D 8F 02	STA \$028F	
C4AA	AD 8F 02	LDA \$028F	
C4AD	D0 28	BNE \$C4D7	
C4AF	CE 8C 02	DEC \$028C	

# Anatomy of the 1541 Disk Drive

C4B2	10	DE	BPL	\$C492	
C4B4	60		RTS		
C4B5	20	17	C6	JSR	\$C617
C4B8	F0	F0		BEO	\$C4AA
C4BA	20	D8	C4	JSR	\$C4D8
C4BD	AE	53	02	LDX	\$0253
C4C0	10	07		BPL	\$C4C9
C4C2	AD	8F	02	LDA	\$028F
C4C5	F0	EE		BEO	\$C4B5
C4C7	D0	0E		BNE	\$C4D7
					yes
					no, then done
C4C9	AD	96	02	LDA	\$0296
C4CC	F0	09		BEO	\$C467
C4CE	B5	E7		LDA	\$E7,X
C4D0	29	07		AND	#\$07
C4D2	CD	96	02	CMP	\$0296
C4D5	D0	DE		BNE	\$C4B5
C4D7	60			RTS	
					same as desired file type?
					no
C4D8	A2	FF		LDX	#\$FF
C4DA	8E	53	02	STX	\$0253
C4DD	E8			INX	
C4DE	8E	8A	02	STX	\$028A
C4E1	20	89	C5	JSR	\$C589
C4E4	F0	06		BEO	\$C4EC
C4E6	60			RTS	
					set pointer to data
C4E7	20	94	C5	JSR	\$C594
C4EA	D0	FA		BNE	\$C4E6
C4EC	A5	7F		LDA	\$7F
C4EE	55	E2		EOR	\$E2,X
C4F0	4A			LSR	A
C4F1	90	0B		BCC	\$C4FE
C4F3	29	40		AND	#\$40
C4F5	F0	F0		BEO	\$C4E7
C4F7	A9	02		LDA	#\$02
C4F9	CD	8C	02	CMP	\$028C
C4FC	F0	E9		BEO	\$C4E7
C4FE	BD	7A	02	LDA	\$027A,X
C501	AA			TAX	
C502	20	A6	C6	JSR	\$C6A6
C505	A0	03		LDY	#\$03
C507	4C	1D	C5	JMP	\$C51D
					search both drives?
					yes
C50A	BD	00	02	LDA	\$0200,X
C50D	D1	94		CMP	(\$94),Y
C50F	F0	0A		BEO	\$C51B
C511	C9	3F		CMP	#\$3F
C513	D0	D2		BNE	\$C4E7
C515	B1	94		LDA	(\$94),Y
C517	C9	A0		CMP	#\$A0
C519	F0	CC		BEO	\$C4E7
C51B	E8			INX	
C51C	C8			INX	
					get chars out of command line
					same character in directory?
					yes
					'?'
					no
					shift blank, end of name?
					yes
					increment pointer



# Anatomy of the 1541 Disk Drive

C51D	EC 76 02	CPX \$0276	end of the name in the command?
C520	B0 09	BCS \$C52B	yes
C522	BD 00 02	LDA \$0200,X	next character
C525	C9 2A	CMP #\$2A	'*'
C527	F0 0C	BEQ \$C535	yes, file found
C529	DO DF	BNE \$C50A	continue search
C52B	C0 13	CPY #\$13	19
C52D	B0 06	BCS \$C535	reached end of name
C52F	B1 94	LDA (\$94),Y	
C531	C9 A0	CMP #\$A0	shift blank, end of name
C533	D0 B2	BNE \$C4E7	not found
C535	AE 79 02	LDX \$0279	
C538	8E 53 02	STX \$0253	
C53B	B5 E7	LDA \$E7,X	
C53D	29 80	AND #\$80	
C53F	8D 8A 02	STA \$028A	
C542	AD 94 02	LDA \$0294	
C545	95 DD	STA \$DD,X	
C547	A5 81	LDA \$81	sector number of the directory
C549	95 D8	STA \$D8,X	enter in table
C54B	A0 00	LDY #\$00	
C54D	B1 94	LDA (\$94),Y	file type
C54F	C8	INY	
C550	48	PHA	
C551	29 40	AND #\$40	isolate scratch-protect bit
C553	85 6F	STA \$6F	(6) and save
C555	68	PLA	
C556	29 DF	AND #\$DF	erase bit 7
C558	30 02	BMI \$C55C	
C55A	09 20	ORA #\$20	set bit 5
C55C	29 27	AND #\$27	erase bits 3 and 4
C55E	05 6F	ORA \$6F	get bit 6 again
C560	85 6F	STA \$6F	
C562	A9 80	LDA #\$80	
C564	35 E7	AND \$E7,X	isolate flag for wildcard
C566	05 6F	ORA \$6F,X	
C568	95 E7	STA \$E7,X	write in table
C56A	B5 E2	LDA \$E2,X	
C56C	29 80	AND #\$80	
C56E	05 7F	ORA \$7F	drive number
C570	95 E2	STA \$E2,X	
C572	B1 94	LDA (\$94),Y	
C574	9D 80 02	STA \$0280,X	first track of file
C577	C8	INY	
C578	B1 94	LDA (\$94),Y	
C57A	9D 85 02	STA \$0285,X	get sector from directory
C57D	AD 58 02	LDA \$0258	record length
C580	D0 07	BNE \$C589	
C582	A0 15	LDY #\$15	
C584	B1 94	LDA (\$94),Y	record length
C586	8D 58 02	STA \$0258	get from directory
C589	A9 FF	LDA #\$FF	
C58B	8D 8F 02	STA \$028F	
C58E	AD 78 02	LDA \$0278	

# Anatomy of the 1541 Disk Drive

C591	8D 79 02	STA \$0279	
C594	CE 79 02	DEC \$0279	
C597	10 01	BPL \$C59A	
C599	60	RTS	
C59A	AE 79 02	LDX \$0279	
C59D	B5 E7	LDA \$E7,X	wildcard flag set?
C59F	30 05	BMI \$C5A6	yes
C5A1	BD 80 02	LDA \$0280,X	track number already set
C5A4	D0 EF	BNE \$C594	yes
C5A6	A9 00	LDA #\$00	
C5A8	8D 8F 02	STA \$028F	
C5AB	60	RTS	
C5AC	A0 00	LDY #\$00	
C5AE	8C 91 02	STY \$0291	
C5B1	88	DEY	
C5B2	8C 53 02	STY \$0253	
C5B5	AD 85 FE	LDA \$FE85	18, directory track
C5B8	85 80	STA \$80	
C5BA	A9 01	LDA #\$01	
C5BC	85 81	STA \$81	sector 1
C4BE	8D 93 02	STA \$0293	
C5C1	20 75 D4	JSR \$D475	read sector
C5C4	AD 93 02	LDA \$0293	
C5C7	D0 01	BNE \$C5CA	
C5C9	60	RTS	
C5CA	A9 07	LDA #\$07	
C5CC	8D 95 02	STA \$0295	number of directory entries (-1)
C5CF	A9 00	LDA #\$00	
C5D1	20 F6 D4	JSR \$D4F6	get pointer from buffer
C5D4	8D 93 02	STA \$0293	save as track number
C5D7	20 E8 D4	JSR \$D4E8	set buffer pointer
C5DA	CE 95 02	DEC \$0295	decrement counter
C5DD	A0 00	LDY #\$00	
C5DF	B1 94	LDA (\$94),Y	first byte from directory
C5E1	D0 18	BNE \$C5FB	
C5E3	AD 91 02	LDA \$0291	
C5E6	D0 2F	BNE \$C617	
C5E8	20 3B DE	JSR \$DE3B	get track and sector number
C5EB	A5 81	LDA \$81	
C5ED	8D 91 02	STA \$0291	sector number
C5F0	A5 94	LDA \$94	
C5F2	AE 92 02	LDX \$0292	
C5F5	8D 92 02	STA \$0292	buffer pointer
C5F8	F0 1D	BEQ \$C617	
C5FA	60	RTS	
C5FB	A2 01	LDX #\$01	
C5FD	EC 92 02	CPX \$0292	buffer pointer to one?
C600	D0 2D	BNE \$C62F	
C602	F0 13	BEQ \$C617	
C604	AD 85 FE	LDA \$FE85	18, track number of BAM,

# Anatomy of the 1541 Disk Drive

C607	85 80	STA \$80	track number
C609	AD 90 02	LDA \$0290	
C60C	85 81	STA \$81	sector number
C60E	20 75 D4	JSR \$D475	read block
C611	AD 94 02	LDA \$0294	
C614	20 C8 D4	JSR \$D4C8	set buffer pointer
C617	AD FF	LDA #\$FF	
C619	8D 53 02	STA \$0253	erase-file found flag
C61C	AD 95 02	LDA \$0295	
C61F	30 08	BMI \$C629	all directory entries checked?
C621	A9 20	LDA #\$20	
C623	20 C6 D1	JSR \$D1C6	inc buffer ptr by 32, next entry
C626	4C D7 C5	JMP \$C567	and continue
C629	20 4D D4	JSR \$D44D	set buffer pointer
C62C	4C C4 C5	JMP \$C5C4	read next block
C62F	A5 94	LDA \$94	
C631	8D 94 02	STA \$0294	
C634	20 3B DE	JSR \$DE3B	get track & sector no. from buffer
C637	A5 81	LDA \$81	
C639	8D 90 02	STA \$0290	save sector number
C63C	60	RTS	

\*\*\*\*\* test and initialize drive

C63D	A5 68	LDA \$68	
C63F	D0 28	BNE \$C669	
C641	A6 7F	LDX \$7F	drive number
C643	56 1C	LSR \$1C,X	disk changed?
C645	90 22	BCC \$C669	no, then done
C647	A9 FF	LDA \$FF	
C649	8D 98 02	STA \$0298	set error flag
C64C	20 0E D0	JSR \$D00E	read directory track
C64F	A0 FF	LDY #\$FF	
C651	C9 02	CMP #\$02	20, 'read error'?
C653	F0 0A	BEQ \$C65F	yes
C655	C9 03	CMP #\$03	21, 'read error'?
C657	F0 06	BEQ \$C65F	yes
C659	C9 0F	CMP #\$0F	74, 'drive not ready'?
C65B	F0 02	BEQ \$C65F	yes
C65D	A0 00	LDY #\$00	
C65F	A6 7F	LDX \$7F	drive number
C661	98	TYA	
C662	95 FF	STA \$FF,X	save error flag
C664	D0 03	BNE \$C669	error?
C666	20 42 D0	JSR \$D042	load BAM
C669	A6 7F	LDX \$7F	drive number
C66B	B5 FF	LDA \$FF,X	transmit error code
C66D	60	RTS	

\*\*\*\*\* name of file in directory buffer

C66E	48	PHA	
C66F	20 A6 C6	JSR \$C6A6	get end of the name
C672	20 88 C6	JSR \$C688	write filename in buffer
C675	68	PLA	

# Anatomy of the 1541 Disk Drive

C676	38	SEC	
C677	ED 4B 02	SBC \$024B	compare len with max length
C67A	AA	TAX	
C67B	F0 0A	BEQ \$C687	
C67D	90 08	BCC \$C687	
C67F	A9 A0	LDA #\$A0	pad with 'Shift blank'
C681	91 94	STA (\$94),Y	
C683	C8	INY	
C684	CA	DEX	
C685	D0 FA	BNE \$C681	
C687	60	RTS	

\*\*\*\*\*

C688	98	TYA	buffer number
C689	0A	ASL A	
C68A	A8	TAY	times 2 as pointer
C68B	B9 99 00	LDA \$0099,Y	
C68E	85 94	STA \$94	
C690	B9 9A 00	LDA \$009A	buffer pointer after \$94/\$95
C693	85 95	STA \$95	
C695	A0 00	LDY #\$00	
C697	BD 00 02	LDA \$0200,X	transmit characters in buffer
C69A	91 94	STA (\$94),Y	
C69C	C8	INY	
C69D	F0 06	BEQ \$C6A5	buffer already full?
C69F	E8	INX	
C6A0	EC 76 02	CPX \$0276	
C6A3	90 F2	BCC \$C697	
C6A5	60	RTS	

\*\*\*\*\* search for end of name in command

C6A6	A9 00	LDA #\$00	
C6A8	8D 4B 02	STA \$024B	
C6AB	8A	TXA	
C6AC	48	PHA	
C6AD	BD 00 02	LDA \$0200,X	get characters out of buffer
C6B0	C9 2C	CMP #\$2C	','
C6B2	F0 14	BEQ \$C6C8	
C6B4	C9 3D	CMP #\$3D	'='
C6B6	F0 10	BEQ \$C6C8	
C6B8	EE 4B 02	INC \$024B	increment length of name
C6BB	E8	INX	
C6BC	A9 0F	LDA #\$0F	15
C6BE	CD 4B 02	CMP \$024B	
C6C1	90 05	BCC \$C6C8	greater?
C6C3	EC 74 02	CPX \$0274	end of input line?
C6C6	90 E5	BCC \$C6AD	
C6C8	8E 76 02	STX \$0276	
C6CB	68	PLA	
C6CC	AA	TAX	pointer to end of name
C6CD	60	RTS	

\*\*\*\*\*

C6CE	A5 83	LDA \$83	
C6D0	48	PHA	secondary address and channel no.

# Anatomy of the 1541 Disk Drive

C6D1	A5 82	LDA \$82	
C6D3	48	PHA	
C6D4	20 DE C6	JSR \$C6DE	create file entry for directory
C6D7	68	PLA	
C6D8	85 82	STA \$82	
C6DA	68	PLA	get data back
C6DB	85 83	STA \$83	
C6DD	60	RTS	

\*\*\*\*\*

C6DE	A9 11	LDA #\$11	17
C6E0	85 83	STA \$83	secondary address
C6E2	20 EB D0	JSR \$D0EB	open channel to read
C6E5	20 E8 D4	JSR \$D4E8	set buffer pointer
C6E8	AD 53 02	LDA \$0253	
C6EB	10 0A	BPL \$C6F7	not yet last entry?
C6ED	AD 8D 02	LDA \$028D	
C6F0	D0 0A	BNE \$C6FC	
C6F2	20 06 C8	JSR \$C806	write 'blocks free.'
C6F5	18	CLC	
C6F6	60	RTS	
C6F7	AD 8D 02	LDA \$028D	
C6FA	F0 1F	BEQ \$C71B	
C6FC	CE 8D 02	DEC \$028D	
C6FF	D0 0D	BNE \$C70E	
C701	CE 8D 02	DEC \$028D	
C704	20 8F C3	JSR \$C38F	change drive
C707	20 06 C8	JSR \$C806	write 'blocks free.'
C70A	38	SEC	
C708	4C 8F C3	JMP \$C38F	change drive
C70E	A9 00	LDA #\$00	
C710	8D 73 02	STA \$0273	drive no. for header, hi-byte
C713	8D 8D 02	STA \$028D	
C716	20 B7 C7	JSR \$C7B7	write header
C719	38	SEC	
C71A	60	RTS	
C71B	A2 18	LDX #\$18	
C71D	A0 1D	LDY #\$1D	
C71F	B1 94	LDA (\$94),Y	number of blocks hi
C721	8D 73 02	STA \$0273	in buffer
C724	F0 02	BEQ \$C728	zero?
C726	A2 16	LDX #\$16	
C728	88	DEY	
C729	B1 94	LDA (\$94),Y	number of blocks lo
C72B	8D 72 02	STA \$0272	in buffer
C72E	E0 16	CPX #\$16	
C730	F0 0A	BEQ \$C73C	
C732	C9 0A	CMP #\$0A	10
C734	90 06	BCC \$C73C	
C736	CA	DEX	
C737	C9 64	CMP #\$64	100
C739	90 01	BCC \$C73C	
C73B	CA	DEX	

C73C	20 AC C7	JSR \$C7AC	erase buffer
C73F	B1 94	LDA (\$94),Y	file type
C741	48	PHA	
C742	0A	ASL A	bit 7 in carry
C743	10 05	BPL \$C74A	bit 6 not set?
C745	A9 3C	LDA #\$3C	'<' for protected file
C747	9D B2 02	STA \$02B2,X	write behind file type
C74A	68	PLA	
C74B	29 0F	AND #\$0F	isolate bits 0-3
C74D	A8	TAY	as file type marker
C74E	B9 C5 FE	LDA \$FEC5,Y	3rd letter of the file type
C751	9D B1 02	STA \$02B1,X	in buffer
C754	CA	DEX	
C755	B9 C0 FE	LDA \$FEC0,Y	2nd letter of file type
C758	9D B1 02	STA \$02B1,X	in buffer
C75B	CA	DEX	
C75C	B9 BB FE	LDA \$FEBB,Y	1st letter of file type
C75F	9D B1 02	STA \$02B1,X	in buffer
C762	CA	DEX	
C763	CA	DEX	
C764	B0 05	BCS \$C76B	file not closed?
C766	A9 2A	LDA #\$2A	'*'
C768	9D B2 02	STA \$02B2,X	before file type in buffer
C76B	A9 A0	LDA #\$A0	pad with 'shift blank'
676D	9D B1 02	STA \$02B1,X	in buffer
C770	CA	DEX	
C771	A0 12	LDY #\$12	
C773	B1 94	LDA (\$94),Y	filenames
C775	9D B1 02	STA \$02B1,X	write in buffer
C778	CA	DEX	
C779	88	DEY	
C77A	C0 03	CPY #\$03	
C77C	B0 F5	BCS \$C773	
C77E	A9 22	LDA #\$22	'='
C780	9D B1 02	STA \$02B1,X	write before file type
C783	E8	INX	
C784	E0 20	CPX #\$20	
C786	B0 0B	BCS \$C793	
C788	BD B1 02	LDA \$02B1,X	character from buffer
C78B	C9 22	CMP #\$22	'='?
C78D	F0 04	BEO \$C793	
C7BF	C9 A0	CMP #\$A0	'shift blank' at end of name
C791	D0 F0	BNE \$C783	
C793	A9 22	LDA #\$22	fill through '='
C795	9D B1 02	STA \$02B1,X	
C798	E8	INX	
C799	E0 20	CPX #\$20	
C89B	B0 0A	BCS \$C7A7	
C79D	A9 7F	LDA #\$7F	bit 7
C79F	3D B1 02	AND \$02B1,X	
C7A2	9D B1 02	STA \$02B1,X	erase in the remaining chars
C7A5	10 F1	BPL \$C798	
C7A7	20 B5 C4	JSR \$C4B5	search for next directory entry
C7AA	38	SEC	
C7AB	60	RTS	

# Anatomy of the 1541 Disk Drive

```

*****
C7AC    A0 1B      LDY #$1B
C7AE    A9 20      LDA #$20
C7B0    99 B0 02   STA $02B0,Y
C7B3    88         DEY
C7B4    D0 FA      BNE $C7B0
C7B6    60         RTS

*****
C7B7    20 19 F1   JSR $F119
C7BA    20 DF F0   JSR $F0DF
C7BD    20 AC C7   JSR $C7AC
C7C0    A9 FF      LDA #$FF
C7C2    85 6F      STA $6F
C7C4    A6 7F      LDX $7F
C7C6    8E 72 02   STX $0272
C7C9    A9 00      LDA #$00
C7CB    8D 73 02   STA $0273
C7CE    A6 F9      LDX $F9
C7D0    BD E0 FE   LDA $FEE0,X
C7D3    85 95      STA $95
C7D5    AD 88 FE   LDA $FE88
C7D8    85 94      STA $94
C7DA    A0 16      LDY #$16
C7DC    B1 94      LDA ($94),Y
C7DE    C9 A0      CMP #$A0
C7E0    D0 0B      BNE $C7ED
C7E2    A9 31      LDA #$31
C7E4    2C         .BYTE $2C
C7E5    B1 94      LDA ($94),Y
C7E7    C9 A0      CMP #$A0
C7E9    D0 02      BNE $C7ED
C7EB    A9 20      LDA #$20
C7ED    99 B3 02   STA $02B3
C7F0    88         DEY
C7F1    10 F2      BPL $C7E5
C7F3    A9 12      LDA #$12
C7F5    8D B1 02   STA $02B1
C7F8    A9 22      LDA #$22
C7FA    8D B2 02   STA $02B2
C7FD    8D C3 02   STA $02C3
C800    A9 20      LDA #$20
C802    8D C4 02   STA $02C4
C805    60         RTS

*****
C806    20 AC C7   JSR $C7AC
C809    A0 0B      LDY #$0B
C80B    B9 17 C8   LDA $C817,Y
C80E    99 B1 02   STA $02B1,Y
CB11    88         DEY
C812    10 F7      BPL $C80B
C814    4C 4D EF   JMP $EF4D

*****
C806    20 AC C7   JSR $C7AC
C809    A0 0B      LDY #$0B
C80B    B9 17 C8   LDA $C817,Y
C80E    99 B1 02   STA $02B1,Y
CB11    88         DEY
C812    10 F7      BPL $C80B
C814    4C 4D EF   JMP $EF4D

```

erase directory buffer

' ' blank  
write in buffer

create header with disk name  
initialize if needed  
read disk name  
erase buffer

drive number  
as block no. 10 in buffer

block number 10  
buffer number  
hi-byte of the buffer address

\$90, position of disk name  
save

pad buffer with 'shift blank'

'1'

character from buffer  
compare with 'shift blank'

' ' blank  
in buffer

'RVS ON'  
in buffer  
''

write before  
and after disk name  
' ' blank  
behind it

create last line  
erase buffer  
12 characters  
'blocks free.'  
write in buffer

number of free blocks in front

# Anatomy of the 1541 Disk Drive

\*\*\*\*\*

C817 42 4C 4F 43 4B 53 20 46 'blocks f'  
C81F 52 45 45 2E 'ree.'

\*\*\*\*\*

C823	20	98	C3	JSR	\$C398	S command 'scratch'
C826	20	20	C3	JSR	\$C320	ascertain file type
C829	20	CA	C3	JSR	\$C3CA	get drive number
C82C	A9	00		LDA	#\$00	initialize drive if needed
C82E	85	86		STA	\$86	counter for erased files
C830	20	9D	C4	JSR	\$C49D	search for file in directory
C833	30	3D		BMI	\$C872	not found?
C835	20	B7	DD	JSR	\$DDB7	is file open
C838	90	33		BCC	\$C86D	yes
C83A	A0	00		LDY	#\$00	
C83C	B1	94		LDA	(\$94),Y	file type
C83E	29	40		AND	#\$40	scratch protect
C840	D0	2B		BNE	\$C86D	yes
C842	20	B6	C8	JSR	\$C8B6	erase file and note in directory
C845	A0	13		LDY	#\$13	
C847	B1	94		LDA	(\$94),Y	track no. of the first side-sector
C849	F0	0A		BEQ	\$C855	none present?
C84B	85	80		STA	\$80	note track number
C84D	C8			INY		
C84E	B1	94		LDA	(\$94),Y	and sector number
C850	85	81		STA	\$81	
C852	20	7D	C8	JSR	\$C87D	erase side-sector
C855	AE	53	02	LDX	\$0253	file number
C858	A9	20		LDA	#\$20	
C85A	35	E7		AND	#\$E7,X	bit 5 set?
C85C	D0	0D		BNE	\$C86B	yes, file not closed
C85E	BD	80	02	LDA	\$0280,X	get track
C861	85	80		STA	\$80	
C863	BD	85	02	LDA	\$0285,X	and sector
C866	85	81		STA	\$81	
C868	20	7D	C8	JSR	\$C87D	erase file
C86B	E6	86		INC	\$86	increment number of erased files
C86D	20	8B	C4	JSR	\$C48B	search for next file
C870	10	C3		BPL	\$C835	if present, erase
C872	A5	86		LDA	\$86	number of erased files
C874	85	80		STA	\$80	save as 'track'
C876	A9	01		LDA	#\$01	1 as disk status
C878	A0	00		LDY	#\$00	0 as 'sector'
C87A	4C	A3	C1	JMP	\$C1A3	message 'files scratched'

\*\*\*\*\*

C87D	20	5F	EF	JSR	\$EF5F	erase file
C880	20	75	D4	JSR	\$D475	free block in BAM
C883	20	19	F1	JSR	\$F119	get buffer number in BAM
C886	B5	A7		LDA	\$A7,X	
C888	C9	FF		CMP	#\$FF	
C88A	F0	08		BEQ	\$C894	
C88C	AD	F9	02	LDA	\$02F9	
C88F	09	40		ORA	#\$40	
C891	8D	F9	02	STA	\$02F9	



## Anatomy of the 1541 Disk Drive

C894	A9 00	LDA #\$00	
C896	20 C8 D4	JSR \$D4C8	buffer pointer to zero
C899	20 56 D1	JSR \$D156	get track
C89C	85 80	STA \$80	
C89E	20 56 D1	JSR \$D156	get sector
C8A1	85 81	STA \$81	
C8A3	A5 80	LDA \$80	track number
C8A5	D0 06	BNE \$C8AD	not equal to zero
C8A7	20 F4 EE	JSR \$EEF4	write BAM
C8AA	4C 27 D2	JMP \$D227	close channel

C8AD	20 5F EF	JSR \$EE5F	free block in BAM
C8B0	20 4D D4	JSR \$D44D	read next block
C8B3	4C 94 C8	JMP \$C894	and continue

\*\*\*\*\* erase directory entry

C8B6	A0 00	LDY #\$00	
C8B8	98	TYA	
C8B9	91 94	STA (\$94),Y	set file type to zero
C8BB	20 5E DE	JSR \$DE5E	write block
C8BE	4C 99 D5	JMP \$D599	and check

\*\*\*\*\* D-command 'backup'

C8C1	A9 31	LDA #\$31	
C8C3	4C C8 C1	JMP \$C1C8	31, 'syntax error'

\*\*\*\*\* format diskette

C8C6	A9 4C	LDA #\$4C	JMP-command
C8C8	8D 00 06	STA \$0600	
C8CB	A9 C7	LDA #\$C7	
C8CD	8D 01 06	STA \$0601	JMP \$FAC7 in \$600 to \$602
C8D0	A9 FA	LDA #\$FA	
C8D2	8D 02 06	STA \$0602	
C8D5	A9 03	LDA #\$03	
C8D7	20 D3 D6	JSR \$D6D3	set track and sector number
C8DA	A5 7F	LDA \$7F	drive number
C8DC	09 E0	ORA #\$E0	command code for formatting
C8DE	85 03	STA \$03	transmit
C8E0	A5 03	LDA \$03	
C8E2	30 FC	BMI \$C8E0	wait until formatting done
C8E4	C9 02	CMP #\$02	
C8E6	90 07	BCC \$C8EF	smaller than two, then ok
C8E8	A9 03	LDA #\$03	
C8EA	A2 00	LDX #\$00	
C8EC	4C 0A E6	JMP \$E60A	21, 'read error'
C8EF	60	RTS	

\*\*\*\*\* C-command 'copy'

C8F0	A9 E0	LDA #\$E0	
C8F2	8D 4F 02	STA \$024F	
C8F5	20 D1 F0	JSR \$F0D1	
C8F8	20 19 F1	JSR \$F119	get buffer number of BAM
C8FB	A9 FF	LDA #\$FF	
C8FD	95 A7	STA \$A7,X	
C8FF	A9 0F	LDA #\$0F	

# Anatomy of the 1541 Disk Drive

C901	8D 56 02	STA \$0256	
C904	20 E5 C1	JSR \$C1E5	check input line
C907	D0 03	BNE \$C90C	
C909	4C C1 C8	JMP \$C8C1	31, 'syntax error'
C90C	20 F8 C1	JSR \$C1F8	check input
C90F	20 20 C3	JSR \$C320	test drive number
C912	AD 8B 02	LDA \$028B	flag for syntax check
C915	29 55	AND #\$55	
C917	D0 0F	BNE \$C928	
C919	AE 7A 02	LDX \$027A	
C91C	BD 00 02	LDA \$0200,X	character of the command
C91F	C9 2A	CMP #\$2A	'**'
C921	D0 05	BNE \$C928	
C923	A9 30	LDA #\$30	
C925	4C C8 C1	JMP \$C1C8	30, 'syntax error'
C928	AD 8B 02	LDA \$028B	syntax flag
C92B	29 D9	AND #\$D9	
C92D	D0 F4	BNE \$C923	30, 'syntax error'
C92F	4C 52 C9	JMP \$C952	
C932	A9 00	LDA #\$00	
C934	8D 58 02	STA \$0258	
C937	8D 8C 02	STA \$028C	number of drives
C93A	8D 80 02	STA \$0280	track number in directory
C93D	8D 81 02	STA \$0281	
C940	A4 E3	LDA \$E3	
C942	29 01	AND #\$01	
C944	85 7F	STA \$7F	drive number
C946	09 01	ORA #\$01	
C948	8D 91 02	STA \$0291	
C94B	AD 7B 02	LDA \$027B	
C94E	8D 7A 02	STA \$027A	
C951	60	RTS	
C952	20 4F C4	JSR \$C44F	search for file in directory
C955	AD 78 02	LDA \$0278	number of filenames in command
C958	C9 03	CMP #\$03	smaller than three?
C95A	90 45	BCC \$C9A1	yes
C95C	A5 E2	LDA \$E2	first drive number
C95E	C5 E3	CMP \$E3	second drive number
C960	D0 3F	BNE \$C9A1	not on same drive?
C962	A5 DD	LDA \$DD	directory block of the 1st file
C964	C5 DE	CMP \$DE	same dir block as second file?
C966	D0 39	BNE \$C9A1	no
C968	A5 D8	LDA \$D8	directory sector of first file
C96A	C5 D9	CMP \$D9	same dir sector as second file?
C96C	D0 33	BNE \$C9A1	no
C96E	20 CC CA	JSR \$CACC	is file present
C971	A9 01	LDA #\$01	
C973	8D 79 02	STA \$0279	
C976	20 FA C9	JSR \$C9FA	
C979	20 25 D1	JSP \$D125	get data type
C97C	F0 04	BEQ \$C982	rel-file?
C97E	C9 02	CMP #\$02	prg-file

# Anatomy of the 1541 Disk Drive

C980	D0 05	BNE \$C987	no
C982	A9 64	LDA #\$64	
C984	20 C8 C1	JSR \$C1C8	64, 'file type mismatch'
C987	A9 12	LDA #\$12	18
C989	85 83	STA \$83	secondary address
C98B	AD 3C 02	LDA \$023C	
C98E	8D 3D 02	STA \$023D	
C991	A9 FF	LDA #\$FF	
C993	8D 3C 02	STA \$023C	
C996	20 2A DA	JSR \$DA2A	prepare append
C999	A2 02	LDX #\$02	
C99B	20 B9 C9	JSR \$C9B9	copy file
C99E	4C 94 C1	JMP \$C194	done
C9A1	20 A7 C9	JSR \$C9A7	copy file
C9A4	4C 94 C1	JMP \$C194	done
C9A7	20 E7 CA	JSR \$CAE7	
C9AA	A4 E2	LDA \$E2	drive no. of first file
C9AC	29 01	AND #\$01	
C9AE	85 7F	STA \$7F	drive number
C9B0	20 86 D4	JSR \$D486	
C9B3	20 E4 D6	JSR \$D6E4	enter file in directory
C9B6	AE 77 02	LDX \$0277	
C9B9	8E 79 02	STX \$0279	
C9BC	20 FA C9	JSR \$C9FA	
C9BF	A9 11	LDA #\$11	17
C9C1	85 83	STA \$83	
C9C3	20 EB D0	JSR \$D0EB	
C9C6	20 25 D1	JSR \$D125	get data type
C9C9	D0 03	BNE \$C9CE	no rel-file?
C9CB	20 53 CA	JSR \$CA53	
C9CE	A9 08	LDA #\$08	
C9D0	85 F8	STA \$F8	
C9D2	4C D8 C9	JMP \$C9D8	
C9D5	20 9B CF	JSR \$CF9B	write byte in buffer
C9D8	20 35 CA	JSR \$CA35	and get byte
C9DB	A9 80	LDA #\$80	
C9DD	20 A6 DD	JSR \$DDA6	test bit 7
C9E0	F0 F3	BEQ \$C9D5	not set?
C9E2	20 25 D1	JSR \$D125	check file type
C9E5	F0 03	BEQ \$C9EA	rel-file?
C9E7	20 9B CF	JSR \$CF9B	get data byte in buffer
C9EA	AE 79 02	LDX \$0279	
C9ED	E8	INX	
C9EE	EC 78 02	CPX \$0278	
C9F1	90 C6	BCC \$C9B9	
C9F3	A9 12	LDA #\$12	18
C9F5	85 83	STA \$83	
C9F7	4C 02 DB	JMP \$DB02	close channel
C9FA	AE 79 02	LDX \$0279	
C9FD	B5 E2	LDA \$E2,X	drive number
C9FF	29 01	AND #\$01	

# Anatomy of the 1541 Disk Drive

CA01	85	7F		STA \$7F	save
CA03	AD	85	FE	LDA \$FE85	18, directory track
CA06	85	80		STA \$80	save
CA08	B5	D8		LDA \$D8,X	directory sector
CA0A	85	81		STA \$81	
CA0C	20	75	D4	JSR \$D475	read block
CA0F	AE	79	02	LDX \$0279	
CA12	B5	DD		LDA \$DD,X	pointer in block
CA14	20	C8	D4	JSR \$D4C8	set buffer pointer
CA17	AE	79	02	LDX \$0279	
CA1A	B5	E7		LDA \$E7,X	file type
CA1C	29	07		AND #\$07	isolate
CA1E	8D	4A	02	STA \$024A	and save
CA21	A9	00		LDA #\$00	
CA23	8D	58	02	STA \$0258	
CA26	20	A0	D9	JSR \$D9A0	get parameters for rel-file
CA29	A0	01		LDY #\$01	
CA2B	20	25	D1	JSR \$D125	get file type
CA2E	F0	01		BEQ \$CA31	rel-file?
CA30	C8			INY	
CA31	98			TYA	
CA32	4C	C8	D4	JMP \$D4C8	set buffer pointer
CA35	A9	11		LDA #\$11	17
CA37	85	83		STA \$83	
CA39	20	9B	D3	JSR \$D39B	open channel and get byte
CA3C	85	85		STA \$85	
CA3E	A6	82		LDX \$82	channel number
CA40	B5	F2		LDA \$F2,X	
CA42	29	08		AND #\$08	isolate end marker
CA44	85	F8		STA \$F8	
CA46	D0	0A		BNE \$CA52	not set?
CA48	20	25	D1	JSR \$D125	get data type
CA4B	F0	05		BEQ \$CA52	rel-file?
CA4D	A9	80		LDA #\$80	
CA4F	20	97	DD	JSR \$DD97	set bit 7
CA52	60			RTS	
CA53	20	D3	D1	JSR \$D1D3	set drive number
CA56	20	CB	E1	JSR \$E1CB	
CA59	A5	D6		LDA \$D6	
CA5B	48			PHA	
CA5C	A5	D5		LDA \$D5	
CA5E	48			PHA	
CA5F	A9	12		LDA #\$12	18
CA61	85	83		STA \$83	
CA63	20	07	D1	JSR \$D107	open write channel
CA66	20	D3	D1	JSR \$D1D3	set drive number
CA69	20	CB	E1	JSR \$E1CB	
CA6C	20	9C	E2	JSR \$E29C	
CA6F	A5	D6		LDA \$D6	
CA71	85	87		STA \$87	
CA73	A5	D5		LDA \$D5	
CA75	85	86		STA \$86	
CA77	A9	00		LDA #\$00	
CA79	85	88		STA \$88	

# Anatomy of the 1541 Disk Drive

```
CA7B 85 D4 STA $D4
CA7D 85 D7 STA $D7
CA7F 68 PLA
CA80 85 D5 STA $D5
CA82 68 PLA
CA83 85 D6 STA $D6
CA85 4C 3B E3 JMP $E33B
```

```
***** R-command, 'rename'
CA88 20 20 C3 JSR $C320 get drive no. from command line
CA8B A5 E3 LDA $E3
CA8D 29 01 AND #$01
CA8F 85 E3 STA $E3 2nd drive number
CA91 C5 E2 CMP $E2 compare with 1st drive number
CA93 F0 02 BEQ $CA97 same?
CA95 09 80 ORA #$80
CA97 85 E2 STA $E2
CA99 20 4F C4 JSR $C44F search for file in directory
CA9C 20 E7 CA JSR $CAE7 does name exist?
CA9F A5 E3 LDA $E3
CAA1 29 01 AND #$01
CAA3 85 7F STA $7F drive number
CAA5 A5 D9 LDA $D9
CAA7 85 81 STA $81 sector number
CAA9 20 57 DE JSR $DE57 read block from directory
CAAC 20 99 D5 JSR $D599 ok?
CAAF A5 DE LDA $DE pointer to directory entry
CAB1 18 CLC
CAB2 69 03 ADC #$03 pointer plus 3 to file name
CAB4 20 C8 D4 JSR $D4C8 set buffer pointer
CAB7 20 93 DF JSR $DF93 get buffer number
CABA A8 TAY
CARB AE 7A 02 LDX $027A
CABE A9 10 LDA #$10 16 characters
CAC0 20 6E C6 JSR $C66E write name in buffer
CAC3 20 5E DE JSR $DE5E write block to directory
CAC6 20 99 D5 JSR $D599 ok?
CAC9 4C 94 C1 JMP $C194 done, prepare disk status
```

```
***** check if file present
CACC A5 E8 LDA $E8 file type
CACE 29 07 AND #$07
CAD0 8D 4A 02 STA $024A save
CAD3 AE 78 02 LDX $0278
CAD6 CA DEX
CAD7 EC 77 02 CPX $0277
CADA 90 0A BCC $CAE6
CADC BD 80 02 LDA $0280,X track number
CADF D0 F5 BNE $CAD6 not zero?
CAE1 A9 62 LDA #$62
CAE3 4C C8 C1 JMP $C1C8 62, 'file not found'
CAE6 60 RTS

CAE7 20 CC CA JSR $CACC does file exist with old name?
CAEA 8D 80 02 LDA $0280,X track number of new file
```

# Anatomy of the 1541 Disk Drive

CAED	F0 05	BEQ \$CAF4	file erased?
CAEF	A9 63	LDA #\$63	
CAF1	4C C8 C1	JMP \$C1C8	63, 'file exists'
CAF4	CA	DEX	
CAF5	10 F3	BPL \$CAEA	
CAF7	60	RTS	

\*\*\*\*\* M-command, 'memory'

CAF8	AD 01 02	LDA \$0201	2nd character from buffer
CAFB	C9 2D	CMP #\$2D	'-'
CAFD	D0 4C	BNE \$CB4B	
CAFF	AD 03 02	LDA \$0203	
CB02	85 6F	STA \$6F	address in \$6F/\$70
CB04	AD 04 02	LDA \$0204	
CB07	85 70	STA \$70	
CB09	A0 00	LDY #\$00	
CB0B	AD 02 02	LDA \$0202	3rd character from buffer
CB0E	C9 52	CMP #\$52	'R'
CB10	F0 0E	BEQ \$CB20	to memory read
CB12	20 58 F2	JSR \$F258	(RTS)
CB15	C9 57	CMP #\$57	'W'
CB17	F0 37	BEQ \$CB50	to memory write
CB19	C9 45	CMP #\$45	'E'
CB1B	D0 2E	BNE \$CB4B	
CB1D	6C 6F 00	JMP (\$006F)	memory-execute

\*\*\*\*\* M-R, 'Memory-Read'

CB20	B1 6F	LDA (\$6F),Y	read byte
CB22	85 85	STA \$85	
CB24	AD 74 02	LDA \$0274	length of command line
CB27	C9 06	CMP #\$06	less than 6?
CB29	90 1A	BCC \$CB45	yes
CB2B	AE 05 02	LDX \$0205	number
CB2E	CA	DEX	
CB2F	F0 14	BEQ \$CB45	only one byte?
CB31	8A	TXA	number of bytes
CB32	18	CLC	
CB33	65 6F	ADC \$6F	plus start address
CB35	E6 6F	INC \$6F	
CB37	8D 49 02	STA \$0249	end pointer
CB3A	A5 6F	LDA \$6F	
CB3C	85 A5	STA \$A5	buffer pointer for error message
CB3E	A5 70	LDA \$70	set to start address for 'M-R'
CB40	85 A6	STA \$A6	
CB42	4C 43 D4	JMP \$D443	byte out
CB45	20 EB D0	JSR \$D0EB	open read channel
CB48	4C 3A D4	JMP \$D43A	byte out
CB4B	A9 31	LDA #\$31	
CB4D	4C C8 C1	JMP \$C1C8	31, 'syntax error'

\*\*\*\*\* M-W, 'memory-write'

CB50	B9 06 02	LDA \$0206,Y	read character
CB53	91 6F	STA (\$6F),Y	and save

# Anatomy of the 1541 Disk Drive

CB55	C8		INY	
CB56	CC 05 02	CPY \$0205	number of characters	
CB59	90 F5	BCC SCB50	all characters?	
CB5B	60	RTS		
*****				
CB5C	AC 01 02	LDY \$0201	U-command, 'user'	
CB5F	C0 30	CPY #\$30	second char	
CB61	D0 09	BNE SCB6C	'0'	
CB63	A9 EA	LDA #\$EA	no	
CB65	85 6B	STA \$6B	ptr to table of user-addresses	
CB67	A9 FF	LDA #\$FF	\$FFEA	
CB69	85 6C	STA \$6C		
CB6B	60	RTS		
CB6C	20 72 CB	JSR SCB72		
CB6F	4C 94 C1	JMP SC194	done, prepare error message	
CB72	88	DEY		
CB73	98	TYA		
CB74	29 0F	AND #\$0F	number	
CB76	0A	ASL A	times 2	
CB77	A8	TAY		
CB78	B1 6B	LDA (\$6B),Y	as pointer in table	
CB7A	85 75	STA \$75		
CB7C	C8	INY	address at \$75/\$76	
CB7D	B1 6B	LDA (\$6B),Y		
CB7F	85 76	STA \$76		
CB81	6C 75 00	JMP (\$0075)	execute function	
*****				
CB84	AD 8E 02	LDA \$028E	open direct access channel, '#'	
CB87	85 7F	STA \$7F	last drive number	
CB89	A5 83	LDA \$83	drive number	
CB8B	48	PHA	channel number	
CB8C	20 3D C6	JSR SC63D	check drive and initialize	
CB8F	68	PLA		
CB90	85 83	STA \$83		
CB92	AE 74 02	LDX \$0274	length of filename	
CB95	CA	DEX		
CB96	D0 0D	BNE SCBA5	greater than one?	
CB98	A9 01	LDA \$01		
CB9A	20 E2 D1	JSR \$D1E2	layout buffer and channel	
CB9D	4C F1 CB	JMP SCBF1	set flags, done	
CBA0	A9 70	LDA #\$70		
CBA2	4C C8 C1	JMP SC1C8	70, 'no channel'	
CBA5	A0 01	LDY \$01		
CBA7	20 7C CC	JSR SCC7C	get buffer number	
CBAA	AE 85 02	LDX \$0285	buffer number	
CBAD	E0 05	CPX \$05	bigger than 5?	
CBAF	B0 EF	BCS SCBA0	70, 'no channel'	
CBB1	A9 00	LDA \$00		
CBB3	85 6F	STA \$6F		
CBB5	85 70	STA \$70		

CBB7	38		SEC	
CBB8	26	6F	ROL \$6F	
CBBA	26	70	ROL \$70	
CBBC	CA		DEX	
CBBD	10	F9	RPL \$CBB8	
CBBF	A5	6F	LDA \$6F	
CBC1	2D	4F 02	AND \$024F	
CBC4	D0	DA	BNE \$CBA0	
CBC6	A5	70	LDA \$70	
CBC8	2D	50 02	AND \$0250	
CBCB	D0	D3	BNE \$CBA0	
CBCD	A5	6F	LDA \$6F	
CBCF	0D	4F 02	ORA \$024F	
CBD2	8D	4F 02	STA \$024F	
CBD5	A5	70	LDA \$70	
CBD7	0D	50 02	ORA \$0250	
CBD9	8D	50 02	STA \$0250	
CBD0	A9	00	LDA #\$00	
CBD1	20	E2 D1	JSR \$D1E2	search channel
CBE2	A6	82	LDX \$82	channel number
CBE4	AD	85 02	LDA \$0285	buffer number
CBE7	95	A7	STA \$A7,X	
CBE9	AA		TAX	
CBEA	A5	7F	LDA \$7F	drive number
CBEC	95	00	STA \$00,X	
CBEE	9D	5B 02	STA \$025B,X	
CBF1	A6	83	LDX \$83	secondary address
CBF3	BD	2B 02	LDA \$022B,X	
CBF6	09	40	ORA #\$40	set READ and WRITE flags
CBF8	9D	2B 02	STA \$022B,X	
CBFB	A4	82	LDY \$82	channel number
CBFD	A9	FF	LDA #\$FF	
CBFF	99	44 02	STA \$0244,Y	end pointer
CC02	A9	89	LDA #\$89	
CC04	99	F2 00	STA \$00F2,Y	set READ and WRITE flags
CC07	B9	A7 00	LDA \$00A7,Y	buffer number
CC0A	99	3E 02	STA \$023E,Y	
CC0D	0A		ASL A	times 2
CC0E	AA		TAX	
CC0F	A9	01	LDA #\$01	
CC11	95	99	STA \$99,X	buffer pointer to one
CC13	A9	0E	LDA #\$0E	
CC15	99	EC 00	STA \$00EC,Y	flag for direct access
CC18	4C	94 C1	JMP \$C194	done

*****				B-command, 'Block'
CC1B	A0	00	LDY #\$00	
CC1D	A0	00	LDX #\$00	
CC1F	A9	2D	LDA #\$2D	'_'
CC21	20	68 C2	JSR \$C268	search for minus sign
CC24	D0	0A	BNE \$CC30	found?
CC26	A9	31	LDA #\$31	
CC28	4C	C8 C1	JMP \$C1C8	31, 'syntax error'



# Anatomy of the 1541 Disk Drive

CC2B	A9 30	LDA #\$30	
CC2D	4C C8 C1	JMP \$C1C8	30, 'syntax error'
CC30	8A	TXA	
CC31	D0 F8	BNE \$CC2B	comma, then error
CC33	A2 05	LDX #\$05	
CC35	B9 00 02	LDA \$0200,Y	char from buffer
CC38	DD 5D CC	CMP \$CC5D,X	compare with 'AFRWEF'
CC3B	F0 05	BEQ \$CC42	found?
CC3D	CA	DEX	
CC3E	10 F8	BPL \$CC38	compare with all characters
CC40	30 E4	BMI \$CC26	not found, error
CC42	8A	TXA	
CC43	09 80	ORA #\$80	command number, set bit 7
CC45	8D 2A 02	STA \$022A	
CC48	20 6F CC	JSR \$CC6F	get parameters
CC4B	AD 2A 02	LDA \$022A	
CC4E	0A	ASL A	number times 2
CC4F	AA	TAX	as index
CC50	BD 64 CC	LDA \$CC64,X	address of command hi
CC53	85 70	STA \$70	
CC55	BD 63 CC	LDA \$CC63,X	address lo
CC58	85 6F	STA \$6F	
CC5A	6C 6F 00	JMP (\$006F)	jump to command
*****			names of the various block cmds
CC5D	41 46 52 57 45 50		'AFRWEF'
*****			addresses of block commands
CC63	03 CD		\$CD03, B-A
CC65	F5 CC		\$CCF5, B-F
CC67	56 CD		\$CD56, B-R
CC69	73 CD		\$CD73, B-W
CC6B	A3 CD		\$CDA3, B-E
CC6D	BD CD		\$CDBD, B-P
*****			get parameters for block commands
CC6F	A0 00	LDY #\$00	
CC71	A2 00	LDX #\$00	
CC73	A9 3A	LDA #\$3A	':'
CC75	20 68 C2	JSR \$C268	test line to colon
CC78	D0 02	BNE \$CC7C	found?
CC7A	A0 03	LDY #\$03	no, begin at 4th character
CC7C	B9 00 02	LDA \$0200,Y	search for separating char
CC7F	C9 20	CMP #\$20	' ' blank
CC81	F0 08	BEQ \$CC8B	
CC83	C9 1D	CMP #\$1D	cursor right
CC85	F0 04	BEQ \$CC8B	
CC87	C9 2C	CMP #\$2C	',' comma
CC89	D0 07	BNE \$CC92	
CC8B	C8	INY	
CC8C	CC 74 02	CPY \$0274	line end?
CC8F	90 EB	BCC \$CC7C	
CC91	60	RTS	

# Anatomy of the 1541 Disk Drive

CC92	20	A1	CC	JSR \$CCA1	preserve next parameter
CC95	EE	77	02	INC \$0277	increment parameter counter
CC98	AC	79	02	LDY \$0279	
CC9B	E0	04		CPX #\$04	compare with maximum number
CC9D	90	EC		BCC \$CC8B	
CC9F	B0	8A		BCS \$CC2B	30, 'syntax error'
CCA1	A9	00		LDA #\$00	
CCA3	85	6F		STA \$6F	
CCA5	85	70		STA \$70	erase storage area for decimal #s
CCA7	85	72		STA \$72	
CCA9	A2	FF		LDX \$FF	
CCAB	B9	00	02	LDA \$0200,Y	get characters from input buffer
CCAE	C9	40		CMP #\$40	
CCB0	B0	18		BCS \$CCCA	no digits?
CCB2	C9	30		CMP #\$30	'0'
CCB4	90	14		BCC \$CCCA	no digits?
CCB6	29	0F		AND #\$0F	convert ASCII digits to hex
CCB8	48			PHA	and save
CCB9	A5	70		LDA \$70	
CCBB	85	71		STA \$71	move digits one further
CCBD	A4	6F		LDA \$6F	
CCBF	85	70		STA \$70	
CCC1	68			PLA	
CCC2	85	6F		STA \$6F	note read number
CCC4	C8			INY	increment pointer in input buffer
CCC5	CC	74	02	CPY \$0274	line end reached
CC07	90	E1		BCC \$CCAB	no
CCCA	8C	79	02	STY \$0279	save pointer
CCCD	18			CLC	
CCCE	A9	00		LDA #\$00	
CCD0	E8			INX	
CCD1	E0	03		CPX #\$03	
CCD3	B0	0F		BCS \$CCE4	convert hex digits to one byte
CCD5	B4	6F		LDY \$6F,X	
CCD7	88			DEY	
CCD8	30	F6		BMI \$CCD0	
CCDA	7D	F2	CC	ADC \$CCF2,X	add decimal value
CCDD	90	F8		BCC \$CCD7	
CCDF	18			CLC	
CCE0	E6	72		INC \$72	
CCE2	D0	F3		BNE \$CCD7	
CCE4	48			PHA	
CCE5	AE	77	02	LDX \$0277	counter for parameters
CCE8	A5	72		LDA \$72	
CCEA	9D	80	02	STA \$0280,X	hi-byte
CCED	68			PLA	
CCEE	9D	85	02	STA \$0285,X	lo-byte
CCF1	60			RTS	

\*\*\*\*\* decimal values  
 CCF2 01 0A 64 1, 10, 100

\*\*\*\*\* B-F command, 'Block-Free'  
 CCF5 20 F5 CD JSR \$CDF5 get track, sector and drive no.  
 CCF8 20 5F EF JSR \$EF5F free block

# Anatomy of the 1541 Disk Drive

CCFB	4C 94 C1	JMP \$C194	done, prepare error message
*****			
CCFE	A9 01	LDA #\$01	
CD00	8D F9 02	STA \$02F9	
*****			
CD03	20 F5 CD	JSR \$CDF5	B-A command, 'Block-Allocate'
CD06	A5 81	LDA \$81	get track, sector and drive no.
CD08	48	PHA	sector
CD09	20 FA F1	JSR \$F1FA	save
CD0C	F0 0B	BEO \$CD19	find block in BAM
CD0E	68	PLA	block allocated?
CD0F	C5 81	CMP \$81	desired sector
CD11	D0 19	BNE \$CD2C	= next free sector?
CD13	20 90 EF	JSR \$EF90	no
CD16	4C 94 C1	JMP \$C194	allocate block in BAM
			done
CD19	68	PLA	
CD1A	A9 00	LDA #\$00	
CD1C	85 81	STA \$81	sector 0
CD1E	E6 80	INC \$80	next track
CD20	A5 80	LDA \$80	track number
CD22	CD D7 FE	CMP \$FED7	36, last track number + 1
CD25	B0 0A	BCS \$CD31	>=, then 'no block'
CD27	20 FA F1	JSR \$F1FA	find free block in next track
CD2A	F0 EE	BEO \$CD1A	not found, check next track
CD2C	A9 65	LDA #\$65	
CD2E	20 45 E6	JSR \$E645	65, 'no block' next free block
CD31	A9 65	LDA #\$65	
CD33	20 C8 C1	JSR \$C1C8	65, 'no block' no more free blocks
*****			
CD36	20 F2 CD	JSR \$CDF2	open channel, set parameters
CD39	4C 60 D4	JMP \$D460	read block from disk
*****			
CD3C	20 2F D1	JSR \$D12F	get byte from buffer
CD3F	A1 99	LDA (\$99,X)	set pointer to buffer
CD41	60	RTS	get byte
*****			
CD42	20 36 CD	JSR \$CD36	read block from disk
CD45	A9 00	LDA #\$00	open channel, read block
CD47	20 C8 D4	JSR \$D4C8	
CD4A	20 3C CD	JSR \$CD3C	set buffer pointer to zero
CD4D	99 44 02	STA \$0244,Y	get a byte from the buffer
CD50	A9 89	LDA \$89	
CD52	99 F2 00	STA \$00F2,Y	set read and write flag
CD55	60	RTS	
*****			
CD56	20 42 CD	JSR \$CD42	B-R command, 'Block-Read'
CD59	20 EC D3	JSR \$D3EC	read block from disk
CD5C	4C 94 C1	JMP \$C194	prepare byte from buffer
			prepare error message

# Anatomy of the 1541 Disk Drive

```

*****
CD5F  20 6F CC   JSR $CC6F      U1 command, sub. for 'Block-Read'
CD62  20 42 CD   JSR $CD42      get parameters of the command
CD65  B9 44 02   LDA $0244,Y    read block from disk
CD68  99 3E 02   STA $023E,Y    end pointer
CD6B  A9 FF      LDA #$FF       save as data byte
CD6D  99 44 02   STA $0244,Y    end pointer to $FF
CD70  4C 94 C1   JMP $C194      done, prepare error message

*****
CD73  20 F2 CD   JSR $CDF2      B-W command, 'Block-Write'
CD76  20 E8 D4   JSR $D4E8      open channel
CD79  A8         TAY            set buffer pointer
CD7A  88         DEY
CD7B  C9 02      CMP #$02       buffer pointer lo less than 2?
CD7D  B0 02      BCS $CD81      no
CD7F  A0 01      LDY #$01
CD81  A9 00      LDA #$00
CD83  20 C8 D4   JSR $D4C8      buffer pointer to zero
CD86  98         TYA
CD87  20 F1 CF   JSR $CFF1      write byte in buffer
CD8A  8A         TXA
CD8B  48         PHA
CD8C  20 64 D4   JSR $D464      write block to disk
CD8F  68         PLA
CD90  AA         TAX
CD91  20 EE D3   JSR $D3EE      get byte from buffer
CD94  4C 94 C1   JMP $C194      done, error message

*****
CD97  20 6F CC   JSR $CC6F      U2, sub for 'Block-Write'
CD9A  20 F2 CD   JSR $CDF2      get command parameters
CD9D  20 64 D4   JSR $D464      open channel
CDA0  4C 94 C1   JMP $C194      and write block to disk
                                   done

*****
CDA3  20 58 F2   JSR $F258      'B-E' command, 'Block-Execute'
CDA6  20 36 CD   JSR $CD36      (RTS)
CDA9  A9 00      LDA #$00      open channel and read block
CDAB  85 6F      STA $6F       address low
CDAD  A6 F9      LDX $F9       buffer number
CDAF  BD E0 FE   LDA $FEE0,X   buffer address high
CDB2  85 70      STA $70
CDB4  20 BA CD   JSR $CDBA      execute routine
CDB7  4C 94 C1   JMP $C194      done
CDBA  6C 6F 00   JMP ($006F)   jump to routine

*****
CDBD  20 D2 CD   JSR $CDD2      'B-P' command, 'Block-Pointer'
CDC0  A5 F9      LDA $F9       open channel, get buffer number
CDC2  0A         ASL A         buffer number
CDC3  AA         TAX          * 2
CDC4  AD 86 02   LDA $0286     as index
CDC7  95 99      STA $99,X     pointer value
                                   save as buffer pointer

```

# Anatomy of the 1541 Disk Drive

CDC9	20 2F D1	JSR \$D12F	prepare a byte in buffer
CDCC	20 EE D3	JSR \$D3EE	for output
CDCF	4C 94 C1	JMP \$C194	done
*****			open channel
CDD2	A6 D3	LDX \$D3	
CDD4	E6 D3	INC \$D3	
CDD6	BD 85 02	LDA \$0285,X	buffer number
CDD9	A8	TAY	
CDDA	88	DEY	
CDDB	88	DEY	
CDDC	C0 0C	CPY #\$0C	buffer number smaller than 14?
CDDE	90 05	BCC \$CDE5	yes
CDE0	A9 70	LDA #\$70	
CDE2	4C C8 C1	JMP \$C1C8	70, 'no channel'
CDE5	85 83	STA \$83	secondary address
CDE7	20 EB D0	JSR \$D0EB	open channel
CDEA	B0 F4	BCS \$CDE0	already allocated, 70 'no channel'
CDEC	20 93 DF	JSR \$DF93	buffer number
CDEF	85 F9	STA \$F9	set
CDF1	60	RTS	
*****			
CDF2	20 D2 CD	JSR \$CDD2	check buffer no. and open channel
CDF5	A6 D3	LDX \$D3	channel number
CDF7	BD 85 02	LDA \$0285,X	buffer address
CDFA	29 01	AND #\$01	
CDFC	85 7F	STA \$7F	drive number
CDFE	BD 87 02	LDA \$0287,X	
CE01	85 81	STA \$81	sector
CE03	BD 86 02	LDA \$0286,X	
CE06	85 80	STA \$80	track
CE08	20 5F D5	JSR \$D55F	track and sector ok?
CE0B	4C 00 C1	JMP \$C100	turn LED on
*****			set pointer for rel-file
CE0E	20 2C CE	JSR \$CE2C	record number * record length
CE11	20 6E CE	JSR \$CE6E	divide by 254
CE14	A5 90	LDA \$90	remainder = pointer in data block
CE16	85 D7	STA \$D7	data pointer
CE18	20 71 CE	JSR \$CE71	divide by 120 = side-sector #
CE1B	E6 D7	INC \$D7	
CE1D	E6 D7	INC \$D7	data ptr + 2 (track/sector ptr)
CE1F	A5 8B	LDA \$8B	result of division
CE21	85 D5	STA \$D5	equals side-sector number
CE23	A5 90	LDA \$90	remainder
CE25	0A	ASL A	times 2
CE26	18	CLC	
CE27	69 10	ADC #\$10	plus 16
CE29	85 D6	STA \$D6	=ptr in side-sector to data block
CE2B	60	RTS	
*****			
CE2C	20 D9 CE	JSR \$CED9	erase work storage

# Anatomy of the 1541 Disk Drive

CE2F	85 92	STA \$92	
CE31	A6 82	LDX \$82	channel number
CE33	B5 B5	LDA \$B5,X	record number lo
CE35	85 90	STA \$90	
CE37	B5 BB	LDA \$BB,X	record number hi
CE39	85 91	STA \$91	
CE3B	D0 04	BNE \$CE41	
CE3D	A5 90	LDA \$90	
CE3F	F0 0B	BEO \$CE4C	record number not zero?
CE41	A5 90	LDA \$90	
CE43	38	SEC	
CE44	E9 01	SBC #\$01	then subtract one
CE46	85 90	STA \$90	
CE48	B0 02	BCS \$CE4C	
CE4A	C6 91	DEC \$91	
CE4C	B5 C7	LDA \$C7,X	record length
CE4E	85 6F	STA \$6F	
CE50	46 6F	LSR \$6F	
CE52	90 03	BCC \$CE57	
CE54	20 ED CE	JSR \$CEED	record number * record length
CE57	20 E5 CE	JSR \$CEE5	shift register left
CE5A	A5 6F	LDA \$6F	
CE5C	D0 F2	BNE \$CE50	
CE5E	A5 D4	LDA \$D4	
CE60	18	CLC	
CE61	65 8B	ASC \$8B	
CE63	85 8B	STA \$8B	
CE65	90 06	BCC \$CE6D	result in \$8B/\$8C/\$8D
CE67	E6 8C	INC \$8C	
CE69	D0 02	BNE \$DE6D	
CE6B	E6 8D	INC \$8D	
CE6D	60	RTS	
*****			
CE6E	A9 FE	LDA #\$FE	divide by 254, calculate block #
CE70	2C	.BYTE \$2C	254
*****			
CE71	A9 78	LDA #\$78	divide by 120, calculate
CE73	85 6F	STA \$6F	side-sector number
CE75	A2 03	LDX #\$03	divisor
CE77	B5 8F	LDA \$8F,X	
CE79	48	PHA	
CE7A	B5 8A	LDA \$8A,X	
CE7C	95 8F	STA \$8F,X	
CE7E	68	PLA	
CE7F	95 8A	STA \$8A,X	
CE81	CA	DEX	
CE82	D0 F3	BNE \$CE77	
CE84	20 D9 CE	JSR \$CED9	erase work storage
CE87	A2 00	LDX #\$00	
CE89	B5 90	LDA \$90,X	
CE8B	95 8F	STA \$8F,X	
CE8D	E8	INX	
CE8E	E0 04	CPX #\$04	

# Anatomy of the 1541 Disk Drive

CE90	90 F7	BCC \$CE89	
CE92	A9 00	LDA #\$00	
CE94	85 92	STA \$92	
CE96	24 6F	BIT \$6F	
CE98	30 09	BMI \$CEA3	
CE9A	06 8F	ASL \$8F	
CE9C	08	PHP	
CE9D	46 8F	LSR \$8F	
CE9F	28	PLP	
CEA0	20 E6 CE	JSR \$CEE6	shift register 1 left
CEA3	20 ED CE	JSR \$CEED	add register 0 to register 1
CEA6	20 E5 CE	JSR \$CEE5	shift register 1 left
CEA9	24 6F	BIT \$6F	
CEAB	30 03	BMI \$CEB0	
CEAD	20 E2 CE	JSR \$CEE2	left-shift register 1 twice
CEB0	A5 8F	LDA \$8F	
CEB2	18	CLC	
CEB3	65 90	ADC \$90	
CEB5	85 90	STA \$90	
CEB7	90 06	BCC \$CEBF	
CEB9	E6 91	INC \$91	
CEBB	D0 02	BNE \$CEBF	
CEBD	E6 92	INC \$92	
CEBF	A5 92	LDA \$92	
CEC1	05 91	ORA \$91	
CEC3	D0 C2	BNE \$CE87	
CEC5	A5 90	LDA \$90	
CEC7	38	SEC	
CEC8	E5 6F	SBC \$6F	quotient in \$8B/\$8C/\$8D
CECA	90 0C	BCC \$CED8	
CECC	E6 8B	INC \$8B	
CECE	D0 06	BNE \$CED6	
CED0	E6 8C	INC \$8C	
CED2	D0 02	BNE \$CED6	
CED4	85 90	STA \$90	remainder in \$90
CED8	60	RTS	
*****			erase work storage
CED9	A9 00	LDA #\$00	
CEDB	85 8B	STA \$8B	
CEDD	85 8C	STA \$8C	
CEDF	85 8D	STA \$8D	
CEE1	60	RTS	
*****			left-shift 3-byte register twice
CEE2	20 E5 CE	JSR \$CEE5	
*****			left-shift 3-byte register once
CEE5	18	CLC	
CEE6	29 90	ROL \$90	
CEE8	26 91	ROL \$91	
CEEA	26 92	ROL \$92	
CEEC	60	RTS	
*****			

CEED	18		CLC	
CEEE	A2	FD	LDX #\$FD	
CEFO	B5	8E	LDA \$8E,X	register \$90/\$91/\$92
CEF2	75	93	ADC \$93,X	add to register \$8B/\$8C/\$8D
CEF4	95	8E	STA \$8E,X	
CEF6	E8		INX	
CEF7	D0	F7	BNE \$CEFO	
CEF9	60		RTS	
CEFA	A2	00	LDX #\$00	
CEFC	8A		TXA	
CEFD	95	FA	STA \$FA,X	
CEFF	E8		INX	
CF00	E0	04	CPX #\$04	
CF02	D0	F8	BNE \$CEFC	
CF04	A9	06	LDA #\$06	
CF06	95	FA	STA \$FA,X	
CF08	60		RTS	
CF09	A0	04	LDY #\$04	
CF0B	A6	82	LDX \$82	channel number
CF0D	B9	FA 00	LDA \$00FA,Y	
CF10	96	FA	STX \$FA,Y	
CF12	C5	82	CMP \$82	channel number
CF14	F0	07	BEQ \$CF1D	
CF16	88		DEY	
CF17	30	E1	BMI \$CEFA	
CF19	AA		TAX	
CF1A	4C	0D CF	JMP \$CF0D	
CF1D	60		RTS	
CF1E	20	09 CF	JSR \$CF09	
CF21	20	B7 DF	JSR \$DFB7	
CF24	D0	46	BNE \$CF6C	
CF26	20	D3 D1	JSR \$D1D3	set drive number
CF29	20	8E D2	JSR \$D28E	
CF2C	30	48	BMI \$CF76	
CF2E	20	C2 DF	JSR \$DFC2	
CF31	A5	80	LDA \$80	track
CF33	48		PHA	
CF34	A5	81	LDA \$81	sector
CF36	48		PHA	
CF37	A9	01	LDA #\$01	
CF39	20	F6 D4	JSR \$D4F6	get byte 1 from buffer
CF3C	85	81	STA \$81	sector
CF3E	A9	00	LDA #\$00	
CF40	20	F6 D4	JSR \$D4F6	get byte 0 from buffer
CF43	85	80	STA \$80	track
CF45	F0	1F	BEQ \$CF66	
CF47	20	25 D1	JSR \$D125	check file type
CF4A	F0	0B	BEQ \$CF57	rel-file?
CF4C	20	AB DD	JSR \$DDAB	
CF4F	D0	06	BNE \$CF57	
CF51	20	8C CF	JSR \$CF8C	
CF54	4C	5D CF	JMP \$CF5D	



# Anatomy of the 1541 Disk Drive

CF57	20 8C CF	JSR \$CF8C	
CF5A	20 57 DE	JSR \$DE57	
CF5D	68	PLA	
CF5E	85 81	STA \$81	get sector
CF60	68	PLA	
CF61	85 80	STA \$80	and track number
CF63	4C 6F CF	JMP \$CF6F	
CF66	68	PLA	
CF67	85 81	STA \$81	get back sector
CF69	68	PLA	
CF6A	85 80	STA \$80	and track number
CF6C	20 8C CF	JSR \$CF8C	
CF6F	20 93 DF	JSR \$DF93	
CF72	AA	TAX	
CF73	4C 99 D5	JMP \$D599	and verify
CF76	A9 70	LDA #\$70	
CF78	4C C8 C1	JMP \$C1C8	70, 'no channel'
CF7B	20 09 CF	JSR \$CF09	
CF7E	20 B7 DF	JSR \$DFB7	
CF81	D0 08	BNE \$CF8B	
CF83	20 8E D2	JSR \$D28E	
CF86	30 EE	BMI \$CF76	
CF88	20 C2 DF	JSR \$DFC2	
CF8B	60	RTS	
*****			change buffer
CF8C	A6 82	LDX \$82	channel number
CF8E	B5 A7	LDA \$A7,X	
CF90	49 80	EOR #\$80	
CF92	95 A7	STA \$A7,X	
CF94	B5 AE	LDA \$AE,X	rotate bit 7 in table
CF96	49 80	EOR #\$80	
CF98	95 AE	STA \$AE,X	
CF9A	60	RTS	
*****			write data byte in buffer
CF9B	A2 12	LDX #\$12	channel 18
CF9D	86 83	STX \$83	
CF9F	20 07 D1	JSR \$D107	open write channel
CFA2	20 00 C1	JSR \$C100	turn LED on
CFA5	20 25 D1	JSR \$D125	check file type
CFA8	90 05	BCC \$CAFA	no rel-file
CAFA	A9 20	LDA #\$20	
CFAC	20 9D DD	JSR \$DD9D	change buffer
CAFA	A5 83	LDA \$83	secondary address
CFB1	C9 0F	CMP #\$0F	15?
CFB3	F0 23	BEQ \$CFD8	yes
CFB5	D0 08	BNE \$CFBF	no
CFB7	A5 84	LDA \$84	secondary address
CFB9	29 8F	AND #\$8F	

# Anatomy of the 1541 Disk Drive

CFBB	C9 0F	CMP #\$0F	greater than 15?
CFBD	B0 19	BCS \$CFD8	then input buffer
CFBF	20 25 D1	JSR \$D125	check file type
CFC2	B0 05	BCS \$CFC9	rel-file or direct access?
CFC4	A5 85	LDA \$85	data byte
CFC6	4C 9D D1	JMP \$D19D	write in buffer
CFC9	D0 03	BNE \$CFCE	direct access file?
CFCB	4C AB E0	JMP \$E0AB	write data byte in rel-file
CFCE	A5 85	LDA \$85	
CFD0	20 F1 CF	JSR \$CFF1	write data byte in buffer
CFD3	A4 82	LDY \$82	channel number
CFD5	4C EE D3	JMP \$D3EE	prepare next byte for output
CFD8	A9 04	LDA #\$04	channel 4
CFDA	85 82	STA \$82	corresponding input buffer
CFDC	20 E8 D4	JSR \$D4E8	set buffer pointer
CFDF	C9 2A	CMP #\$2A	40
CFE1	F0 05	BEQ \$CFE8	buffer end?
CFE3	A5 85	LDA \$85	
CFE5	20 F1 CF	JSR \$CFF1	write data byte in buffer
CFE8	A5 F8	LDA \$F8	end flag set?
CFEA	F0 01	BEQ \$CFED	yes
CFEC	60	RTS	
CFED	EE 55 02	INC \$0255	set command flag
CFF0	60	RTS	
*****			
CFF1	48	PHA	write data byte in buffer
CFF2	20 93 DF	JSR \$DF93	save data byte
CFF5	10 06	BPL \$CFFD	get buffer number
CFF7	68	PLA	associated buffer?
CFF8	A9 61	LDA #\$61	
CFFA	4C C8 C1	JMP \$C1C8	61, 'file not open'
CFFD	0A	ASL A	buffer number times 2
CFFE	AA	TAX	as index
CFFF	68	PLA	data byte
D000	81 99	STA (\$99,X)	write in buffer
D002	F6 99	INC \$99,X	increment buffer pointer
D004	60	RTS	
*****			
D005	20 D1 C1	JSR \$C1D1	I-command, Initialize
D008	20 42 D0	JSR \$D042	find drive number
D00B	4C 94 C1	JMP \$C194	load BAM
*****			
D00E	20 0F F1	JSR \$F10F	prepare disk status
D011	A8	TAY	
D012	B6 A7	LDX \$A7,Y	
D014	E0 FF	CPX #\$FF	
D016	48	PHA	
D019	20 8E D2	JSR \$D28E	

# Anatomy of the 1541 Disk Drive

D01C	AA		TAX	
D01D	A9	70	LDA #\$70	
D021	20	48 E6	JSR \$E648	70, 'no channel'
D024	68		PLA	
D025	A8		TAY	
D026	8A		TXA	
D027	09	80	ORA #\$80	
D029	99	A7 00	STA \$00A7,Y	
D02C	8A		TXA	
D02D	29	0F	AND #\$0F	
D02F	85	F9	STA \$F9	
D031	A2	00	LDA #\$00	
D033	86	81	STX \$81	sector 0
D035	AE	85 FE	LDX \$FE85	18
D038	86	80	STX \$80	track 18
D03A	20	D3 D6	JSR \$D6D3	transmit param to disk controller
D03D	A9	B0	LDA #\$B0	command code 'read block header'
D03F	4C	8C D5	JMP \$D58C	transmit to disk controller

\*\*\*\*\* load BAM

D042	20	D1 F0	JSR \$F0D1	
D045	20	13 D3	JSR \$D313	
D048	20	0E D0	JSR \$D00E	read block
D04B	A6	7F	LDX \$7F	drive number
D04D	A9	00	LDA #\$00	
D04F	9D	51 02	STA \$0251,X	reset flag for 'BAM changed'
D052	8A		TXA	
D053	0A		ASL A	
D054	AA		TAX	
D055	A5	16	LDA \$16	
D057	95	12	STA \$12,X	
D059	A4	17	LDA \$17	save ID
D05B	95	13	STA \$13,X	
D05D	20	86 D5	JSR \$D586	
D060	A5	F9	LDA \$F9	buffer number
D062	0A		ASL A	
D063	AA		TAX	
D064	A9	02	LDA #\$02	buffer pointer to \$200
D066	95	99	STA \$99,X	
D068	A1	99	LDA (\$99,X)	get character from buffer
D06A	A6	7F	LDX \$7F	drive number
D06C	9D	01 01	STA \$0101,X	
D06F	A9	00	LDA #\$00	
D071	95	1C	STA \$1C,X	flag for write protect
D073	95	FF	STA \$FF,X	flag for read error

\*\*\*\*\* calculate blocks free

D075	20	3A EF	JSR \$EF3A	buffer address to \$6D/\$6E
D078	A0	04	LDY #\$04	begin at position 4
D07A	A9	00	LDA #\$00	
D07C	AA		TAX	
D07D	18		CLC	
D07E	71	6D	ADC (\$6D),Y	add no. of free blocks per track
D080	90	01	BCC \$D083	
D082	E8		INX	X as hi-byte

# Anatomy of the 1541 Disk Drive

D083	C8	INY	
D084	C8	INY	plus 4
D085	C8	INY	
D086	C8	INY	
D087	C0 48	CPY #\$48	track 18?
D089	F0 F8	BEO \$D083	then skip
D08B	C0 90	CPY #\$90	last track number?
D08D	D0 EE	BNE \$D07D	no
D08F	48	PHA	lo-byte
D090	8A	TXA	hi-byte
D091	A6 7F	LDX \$7F	drive number
D093	9D FC 02	STA \$02FC,X	hi-byte to \$2FC
D096	68	PLA	lo-byte
D097	9D FA 02	STA \$02FA,X	to \$2FA
D09A	60	RTS	

\*\*\*\*\*

D09B	20 D0 D6	JSR \$D6D0	parameters to disk controller
D09E	20 C3 D0	JSR \$D0C3	read block
D0A1	20 99 D5	JSR \$D599	ok?
D0A4	20 37 D1	JSR \$D137	get byte from buffer
D0A7	85 80	STA \$80	track
D0A9	20 37 D1	JSR \$D137	next byte from buffer
D0AC	85 81	STA \$81	sector
D0AE	60	RTS	

D0AF	20 9B D0	JSR \$D09B	
D0B2	A5 80	LDA \$80	track
D0R4	D0 01	BNE \$D0B7	
D0B6	60	RTS	
D0B7	20 1E CF	JSR \$CF1E	change buffer
D0BA	20 D0 D6	JSR \$D6D0	parameters to disk controller
D0BD	20 C3 D0	JSR \$D0C3	read block
D0C0	4C 1E CF	JMP \$CF1E	change buffer

*****			read block
D0C3	A9 80	LDA #\$80	code for 'read'
D0C5	D0 02	BNE \$D0C9	

*****			write block
D0C7	A9 90	LDA #\$90	code for 'write'
D0C9	8D 4D 02	STA \$024D	save
D0CC	20 93 DF	JSR \$DF93	get buffer number
D0CF	AA	TAX	
D0D0	20 06 D5	JSR \$D506	get track/sector, read/write blk
D0D3	8A	TXA	
D0D4	48	PHA	
D0D5	0A	ASL A	buffer pointer times 2
D0D6	AA	TAX	
D0D7	A9 00	LDA #\$00	
D0D9	95 99	STA \$99,X	pointer in buffer to zero
D0DB	20 25 D1	JSR \$D125	get file type
D0DE	C9 04	CMP #\$04	rel-file or direct access?
D0E0	B0 06	BCS \$D0E8	yes
D0E2	F6 B5	INC \$B5,X	

# Anatomy of the 1541 Disk Drive

```
D0E4  D0 02      BNE $D0E8      increment block counter
D0E6  F6 BB      INC $BB,X
D0E8  68         PLA
D0E9  AA         TAX
D0EA  60         RTS
```

```
***** open channel for reading
D0EB  A5 83      LDA $83      secondary address
D0ED  C9 13      CMP #$13     19
D0EF  90 02      BCC $D0F3    smaller?
D0F1  29 0F      AND #$0F
D0F3  C9 0F      CMP #$0F
D0F5  D0 02      BNE $D0F9
D0F7  A9 10      LDA #$10     16
D0F9  AA         TAX
D0FA  38         SEC
D0FB  BD 2B 02   LDA $022B,X
D0FE  30 06      BMI $D106
D100  29 0F      AND #$0F
D102  85 82      STA $82
D104  AA         TAX
D105  18         CLC          flag for ok
D106  60         RTS
```

```
***** open channel for writing
D107  A4 83      LDA $83      secondary address
D109  C9 13      CMP #$13     19
D10B  90 02      BCC $D10F    smaller?
D10D  29 0F      AND #$0F
D10F  AA         TAX
D110  BD 2B 02   LDA $022B,X  channel number
D113  A8         TAY
D114  0A         ASL A
D115  90 0A      BCC $D121
D117  30 0A      BMI $D123
D119  98         TYA
D11A  29 0F      AND #$0F
D11C  85 82      STA $82
D11E  AA         TAX
D11F  18         CLC          flag for ok
D120  60         RTS

D121  30 F6      BMI $D119
D123  38         SEC          flag for channel allocated
D124  60         RTS
```

```
***** check for file type 'REL'
D125  A6 82      LDX $82
D127  B5 EC      LDA SEC,X
D129  4A         LSR A
D12A  29 07      AND #$07
D12C  C9 04      CMP #$04     'REL'?
D12E  60         RTS
```

```
***** get buffer and channel numbers
```

# Anatomy of the 1541 Disk Drive

D12F	20	93	DF	JSR \$DF93	get buffer number
D132	0A			ASL A	
D133	AA			TAX	
D134	A4	82		LDY \$82	
D136	60			RTS	

*****					get a byte from buffer
D137	20	2F	D1	JSR \$D12F	get buffer and channel number
D13A	B9	44	02	LDA \$0244,Y	end pointer
D13D	F0	12		BEQ \$D151	
D13F	A1	99		LDA (\$99,X)	get byte from buffer
D141	48			PHA	
D142	B5	99		LDA \$99,X	buffer pointer
D144	D9	44	02	CMP \$0244,Y	equal end pointer?
D147	D0	04		BNE \$D14D	no
D149	A9	FF		LDA #\$FF	
D14B	95	99		STA \$99,X	buffer pointer to -1
D14D	68			PLA	data byte
D14E	F6	99		INC \$99,X	increment buffer pointer
D150	60			RTS	
D151	A1	99		LDA (\$99,X)	get character from buffer
D153	F6	99		INC \$99,Y	increment buffer pointer
D155	60			RTS	

*****					get byte and read next block
D156	20	37	D1	JSR \$D137	get byte from buffer
D159	D0	36		BNE \$D191	not last character?
D15B	85	85		STA \$85	save data byte
D15D	B9	44	02	LDA \$0244,Y	end pointer
D160	F0	08		BEQ \$D16A	yes
D162	A9	80		LDA #\$80	
D164	99	F2	00	STA \$00F2,Y	READ-flag
D167	A5	85		LDA \$85	data byte
D169	60			RTS	

D16A	20	1E	CF	JSR \$CF1E	change buffer and read next block
D16D	A9	00		LDA #\$00	
D16F	20	C8	D4	JSR \$D4C8	set buffer pointer to zero
D172	20	37	D1	JSR \$D137	get first byte from buffer
D175	C9	00		CMP #\$00	track number zero
D177	F0	19		BEQ \$D192	yes, then last block
D179	85	80		STA \$80	save last track number
D17B	20	37	D1	JSR \$D137	get next byte
D17E	85	81		STA \$81	save as following track
D180	20	1E	CF	JSR \$CF1E	change buffer and read next block
D183	20	D3	D1	JSR \$D1D3	save drive number
D186	20	D0	D6	JSR \$D6D0	param to disk controller
D189	20	C3	D0	JSR \$D0C3	transmit read command
D18C	20	1E	CF	JSR \$CF1E	change buffer and read block
D18F	A5	85		LDA \$85	get data byte
D191	60			RTS	

D192	20	37	D1	JSR \$D137	get next byte from buffer
D195	A4	82		LDY \$82	
D197	99	44	02	STA \$0244,Y	save as end pointer

# Anatomy of the 1541 Disk Drive

D19A	A5 85	LDA \$85	get data byte back
D19C	60	RTS	
*****			byte in buffer and write block
D19D	20 F1 CF	JSR \$CFF1	byte in buffer
D1A0	F0 01	BEQ \$D1A3	buffer full?
D1A2	60	RTS	
D1A3	20 D3 D1	JSR \$D1D3	get drive number
D1A6	20 1E F1	JSR \$F11E	find free block in BAM
D1A9	A9 00	LDA #\$00	
D1AB	20 C8 D4	JSR \$D4C8	buffer pointer to zero
D1AE	A5 80	LDA \$80	
D1B0	20 F1 CF	JSR \$CFF1	track number as first byte
D1B3	A5 81	LDA \$81	
D1B5	20 F1 CF	JSR \$CFF1	sector number as second byte
D1B8	20 C7 D0	JSR \$D0C7	write block
D1BB	20 1E CF	JSR \$CF1E	change buffer
D1BE	20 D0 D6	JSR \$D6D0	param to disk controller
D1C1	A9 02	LDA #\$02	
D1C3	4C C8 D4	JMP \$D4C8	buffer pointer to 2
*****			increment buffer pointer
D1C6	85 6F	STA \$6F	
D1C8	20 E8 D4	JSR \$D4E8	get buffer pointer
D1CB	18	CLC	
D1CC	65 6F	ADC \$6F	
D1CE	95 99	STA \$99,X	and increment
D1D0	85 94	STA \$94	
D1D2	60	RTS	
*****			get drive number
D1D3	20 93 DF	JSR \$DF93	get drive number
D1D6	AA	TAX	
D1D7	BD 5B 02	LDA \$025B,X	
D1DA	29 01	AND #\$01	isolate drive number
D1DC	85 7F	STA \$7F	and save
D1DE	60	RTS	
*****			find write channel and buffer
D1DF	38	SEC	flag for writing
D1E0	B0 01	BCS \$D1E3	
*****			find read channel and buffer
D1E2	18	CLC	flag for reading
D1E3	08	PHP	save
D1E4	85 6F	STA \$6F	buffer number
D1E6	20 27 D2	JSR \$D227	close channel
D1E9	20 7F D3	JSR \$D37F	allocate free channel
D1EC	85 82	STA \$82	channel number
D1EE	A6 83	LDX \$83	secondary address
D1F0	28	PLP	
D1F1	90 02	BCC \$D1F5	read channel?
D1F3	09 80	ORA #\$80	flag for writing
D1F5	9D 2B 02	STA \$022B,X	set
D1F8	29 3F	AND #\$3F	

# Anatomy of the 1541 Disk Drive

D1FA	A8	TAY	
D1FB	A9 FF	LDA #\$FF	default value
D1FD	99 A7 00	STA \$00A7,Y	
D200	99 AE 00	STA \$00AE,Y	write in associated table
D203	99 CD 00	STA \$00CD,Y	
D206	C6 6F	DEC \$6F	decrement buffer number
D208	30 1C	BMI \$D226	done already?
D20A	20 8E D2	JSR \$D28E	find buffer
D20D	10 08	BPL \$D217	found?
D20F	20 5A D2	JSR \$D25A	erase flags in table
D212	A9 70	LDA #\$70	
D214	4C C8 C1	JMP \$C1C8	70, 'no channel'
D217	99 A7 00	STA \$00A7,Y	buffer number in table
D21A	C6 6F	DEC \$6F	buffer number
D21C	30 08	BMI \$D226	already done?
D21E	20 8E D2	JSR \$D28E	find buffer
D221	30 EC	BMI \$D20F	not found?
D223	99 AE 00	STA \$00AE,Y	buffer number in table
D226	60	RTS	
*****			
D227	A5 83	LDA \$83	close channel
D229	C9 0F	CMP #\$0F	secondary address
D22B	D0 01	BNE \$D22E	15?
D22D	60	RTS	no
			else done already
D22E	A6 83	LDX \$83	
D230	BD 2B 02	LDA \$022B,X	channel number
D233	C9 FF	CMP #\$FF	not associated?
D235	F0 22	BEQ \$D259	then done
D237	29 3F	AND #\$3F	
D239	85 82	STA \$82	channel number
D23B	A9 FF	LDA #\$FF	
D23D	9D 2B 02	STA \$022B,X	erase association in table
D240	A6 82	LDX \$82	
D242	A9 00	LDA #\$00	
D244	95 F2	STA \$F2,X	erase READ and WRITE flag
D246	20 5A D2	JSR \$D25A	free buffer
D249	A6 82	LDX \$82	channel number
D24B	A9 01	LDA #\$01	set bit 0
D24D	CA	DEX	
D24E	30 03	BMI \$D253	shift to correct position
D250	0A	ASL A	
D251	D0 FA	BNE \$D24D	
D253	0D 56 02	ORA \$0256	free in allocation register
D256	8D 56 02	STA \$0256	
D259	60	RTS	
*****			
D25A	A6 82	LDX \$82	free buffer
D25C	B5 A7	LDA \$A7,X	channel number
D25E	C9 FF	CMP #\$FF	buffer number
D260	F0 09	BEQ \$D26B	
D262	48	PHA	not associated?
D263	A9 FF	LDA #\$FF	



# Anatomy of the 1541 Disk Drive

D265	95 A7	STA \$A7,X	erase buffer association
D267	68	PLA	
D268	20 F3 D2	JSR \$D2F3	erase buffer allocation register
D26B	A6 82	LDX \$82	channel number
D26D	B5 AE	LDA \$AE,X	
D26F	C9 FF	CMP #\$FF	associated in second table?
D271	F0 09	BEQ \$D27C	no
D273	48	PHA	
D274	A9 FF	LDA #\$FF	
D276	95 AE	STA \$AE,X	erase association
D278	68	PLA	
D279	20 F3 D2	JSR \$D2F3	erase buffer in allocation reg.
D27C	A6 82	LDX \$82	channel number
D27E	B5 CD	LDA \$CD,X	
D280	C9 FF	CMP #\$FF	associated in 3rd table?
D282	F0 09	BEQ \$D28D	no
D284	48	PHA	
D285	A9 FF	LDA #\$FF	
D287	95 CD	STA \$CD,X	erase association
D289	68	PLA	
D28A	20 F3 D2	JSR \$D2F3	erase buffer in allocation reg
D28D	60	RTS	

\*\*\*\*\* find buffer

D28E	98	TYA	
D28F	48	PHA	
D290	A0 01	LDY #\$01	
D292	20 BA D2	JSR \$D2BA	
D295	10 0C	BPL \$D2A3	
D297	88	DEY	
D298	20 BA D2	JSR \$D2BA	
D29B	10 06	BPL \$D2A3	
D29D	20 39 D3	JSR \$D339	
D2A0	AA	TAX	
D2A1	30 13	BMI \$D2B6	
D2A3	B5 00	LDA \$00,X	
D2A5	30 FC	BMI \$D2A3	
D2A7	A5 7F	LDA \$7F	
D2A9	95 00	STA \$00,X	
D2AB	9D 5B 02	STA \$025B,X	
D2AE	8A	TXA	
D2AF	0A	ASL A	
D2B0	A8	TAY	
D2B1	A9 02	LDA #\$02	
D2B3	99 99 00	STA \$0099,Y	
D2B6	68	PLA	
D2B7	A8	TAY	
D2B8	8A	TXA	
D2B9	60	RTS	
D2BA	A2 07	LDX #\$07	
D2BC	B9 4F 02	LDA \$024F,Y	
D2BF	3D E9 EF	AND \$EFE9,Y	erase bit
D2C2	F0 04	BEQ \$D2C8	
D2C4	CA	DEX	

# Anatomy of the 1541 Disk Drive

D2C5	10	F5		BPL \$D2BC	
D2C7	60			RTS	
D2C8	B9	4F	02	LDA \$024F,Y	
D2CB	5D	E9	EF	EOR \$EFE9,X	rotate bit
D2CE	99	4F	02	STA \$024F,Y	
D2D1	8A			TXA	buffer number
D2D2	88			DEY	
D2D3	30	03		BMI \$D2D8	
D2D5	18			CLC	
D2D6	69	08		ADC #\$08	
D2D8	AA			TAX	buffer number
D2D9	60			RTS	
D2DA	A6	82		LDX \$82	
D2DC	B5	A7		LDA \$A7,X	
D2DE	30	09		BMI \$D2E9	
D2E0	8A			TXA	
D2E1	18			CLC	
D2E2	69	07		ADC #\$07	
D2E4	AA			TAX	
D2E5	B5	A7		LDA \$A7,X	
D2E7	10	F0		BPL \$D2D9	
D2E9	C9	FF		CMP #\$FF	
D2EB	F0	EC		BEQ \$D2D9	
D2ED	48			PHA	
D2EE	A9	FF		LDA #\$FF	
D2F0	95	A7		STA \$A7,X	
D2F2	68			PLA	
D2F3	29	0F		AND #\$0F	
D2F5	A8			TAY	buffer number
D2F6	C8			INY	
D2F7	A2	10		LDX #\$10	16
D2F9	6E	50	02	ROR \$0250	
D2FC	6E	4F	02	ROR \$024F	rotate 16-bit allocation reg.
D2FF	88			DEY	
D300	D0	01		BNE \$D303	
D302	18			CLC	erase bit for buffer
D303	CA			DEX	
D304	10	F3		BPL \$D2F9	
D306	60			RTS	
*****					close all channels
D307	A9	0E		LDA #\$0E	14
D309	85	83		STA \$83	secondary address
D30B	20	27	D2	JSR \$D227	close channel
D30E	C6	83		DEC \$83	next secondary address
D310	D0	F9		BNE \$D30B	
D312	60			RTS	
*****					close channels of other drives
D313	A9	0E		LDA #\$0E	14
D315	85	83		STA \$83	secondary address
D317	A6	83		LDX \$83	
D319	BD	2B	02	LDA \$022B,X	association table
D31C	C9	FF		CMP #\$FF	channel associated?

# Anatomy of the 1541 Disk Drive

D31E	F0 14	BEQ \$D334	no
D320	29 3F	AND #\$3F	
D322	85 82	STA \$82	channel number
D324	20 93 DF	JSR \$DF93	get buffer number
D327	AA	TAX	
D328	BD 5B 02	LDA \$025B,X	drive number
D32B	29 01	AND #\$01	isolate
D32D	C5 7F	CMP \$7F	equal to actual drive number
D32F	D0 03	BNE \$D334	no
D331	20 27 D2	JSR \$D227	close channel
D334	C6 83	DEC \$83	next channel
D336	10 DF	BPL \$D317	
D338	60	RTS	

\*\*\*\*\*

D339	A5 6F	LDA \$6F
D33B	48	PHA
D33C	A0 00	LDY #\$00
D33E	B6 FA	LDX \$FA,Y
D340	B5 A7	LDA \$A7,X
D342	10 04	BPL \$D348
D344	C9 FF	CMP #\$FF
D346	D0 16	BNE \$D35E
D348	8A	TXA
D349	18	CLC
D34A	69 07	ADC #\$07
D34C	AA	TAX
D34D	B5 A7	LDA \$A7,X
D34F	10 04	BPL \$D355
D351	C9 FF	CMP #\$FF
D353	D0 09	BNE \$D35E
D355	C8	INY
D356	C0 05	CPY #\$05
D358	90 E4	BCC \$D33E
D35A	A2 FF	LDX #\$FF
D35C	D0 1C	BNE \$D37A
D35E	86 6F	STX \$6F
D360	29 3F	AND #\$3F
D362	AA	TAX
D363	B5 00	LDA \$00,X
D365	30 FC	BMI \$D363
D367	C9 02	CMP #\$02
D369	90 08	BCC \$D373
D36B	A6 6F	LDX \$6F
D36D	E0 07	CPX #\$07
D36F	90 D7	BCC \$D348
D371	B0 E2	BCS \$D355

D373	A4 6F	LDY \$6F
D375	A9 FF	LDA #\$FF
D377	99 A7 00	STA \$00A7,Y
D37A	68	PLA
D37B	85 6F	STA \$6F
D37D	8A	TXA
D37E	60	RTS

# Anatomy of the 1541 Disk Drive

```

***** find channel and allocate
D37F  A0 00      LDY #$00
D381  A9 01      LDA #$01      set bit 0
D383  2C 56 02   BIT $0256
D386  D0 09      BNE $D391     channel free?
D388  C8         INY
D389  0A         ASL A         rotate bit to left
D38A  D0 F7      BNE $D383     all channels checked?
D38C  A9 70      LDA #$70
D38E  4C C8 C1   JMP $C1C8     70, 'no channel'

D391  49 FF      EOR #$FF     rotate bit model
D393  2D 56 02   AND $0256     erase bit
D396  8D 56 02   STA $0256     allocate channel
D399  98         TYA
D39A  60         RTS

***** get byte for output
D39B  20 EB D0   JSR $D0EB     open channel for reading
D39E  20 00 C1   JSR $C100     turn LED on
D3A1  20 AA D3   JSR $D3AA     get byte in output register
D3A4  A6 82      LDX $82       channel number
D3A6  BD 3E 02   LDA $023E,X   get byte
D3A9  60         RTS

D3AA  A6 82      LDX $82       channel number
D3AC  20 25 D1   JSR $D125     check file type
D3AF  D0 03      BNE $D3B4     no rel-file?
D3B1  4C 20 E1   JMP $E120     get byte from rel-file

D3B4  A5 83      LDA $83       secondary address
D3B6  C9 0F      CMP #$0F      15
D3B8  F0 5A      BEQ $D414     yes, read error channel
D3BA  B5 F2      LDA $F2,X
D3BC  29 08      AND #$08
D3BE  D0 13      BNE $D3D3     end flag set?
D3C0  20 25 D1   JSR $D125     no
D3C3  C9 07      CMP #$07     check file type
D3C5  D0 07      BNE $D3CE     direct access file?
D3C7  A9 89      LDA #$89     no
D3C9  95 F2      STA $F2,X     set READ and WRITE flag
D3CB  4C DE D3   JMP $D3DE

D3CE  A9 00      LDA #$00
D3D0  95 F2      STA $F2,X     erase READ and WRITE flag
D3D2  60         RTS

D3D3  A5 83      LDA $83       secondary address
D3D5  F0 32      BEQ $D409     zero, LOAD?
D3D7  20 25 D1   JSR $D125     check file type
D3DA  C9 04      CMP #$04     rel-file or direct access?
D3DC  90 22      BCC $D400     no
D3DE  20 2F D1   JSR $D12F     get buffer and channel number
D3E1  B5 99      LDA $99,X     buffer pointer

```

# Anatomy of the 1541 Disk Drive

D3E3	D9 44 02	CMP \$0244,Y	equal end pointer?
D3E6	D0 04	BNE \$D3EC	no
D3E8	A9 00	LDA #\$00	
D3EA	95 99	STA \$99,X	buffer pointer to zero
D3EC	F6 99	INC \$99,X	increment buffer pointer
D3EE	A1 99	LDA (\$99,X)	get byte from buffer
D3F0	99 3E 02	STA \$023E,Y	into output register
D3F3	B5 99	LDA \$99,X	buffer pointer
D3F5	D9 44 02	CMP \$0244,Y	equal end pointer?
D3F8	D0 05	BNE \$D3FF	no
D3FA	A9 81	LDA #\$81	
D3FC	99 F2 00	STA \$00F2,Y	set flags
D3FF	60	RTS	
D400	20 56 D1	JSR \$D156	get byte from buffer
D403	A6 82	LDX \$82	channel number
D405	9D 3E 02	STA \$023E,X	byte in output register
D408	60	RTS	
D409	AD 54 02	LDA \$0254	flag for directory?
D40C	F0 F2	BEQ \$D400	no
D40E	20 67 ED	JSR \$ED67	create directory line
D411	4C 03 D4	JMP \$D403	
D414	20 E8 D4	JSR \$D4E8	set buffer pointer
D417	C9 D4	CMP #\$D4	
D419	D0 18	BNE \$D433	
D41B	A5 95	LDA \$95	
D41D	C9 02	CMP #\$02	
D41F	D0 12	BNE \$D433	
D421	A9 0D	LDA #\$0D	CR
D423	85 85	STA \$85	in output register
D425	20 23 C1	JSR \$C123	erase error flags
D428	A9 00	LDA #\$00	
D42A	20 C1 E6	JSR \$E6C1	create 'ok' message
D42D	C6 A5	DEC \$A5	set buffer pointer back
D42F	A9 80	LDA #\$80	set READ flag
D431	D0 12	BNE \$D445	
D433	20 37 D1	JSR \$D137	get byte from buffer
D436	85 85	STA \$85	into output register
D438	D0 09	BNE \$D443	
D43A	A9 D4	LDA #\$D4	
D43C	20 C8 D4	JSR \$D4C8	set buf ptr in front of error ptr
D43F	A9 02	LDA #\$02	
D441	95 9A	STA \$9A,X	hl-address
D443	A9 88	LDA #\$88	set READ flag
D445	85 F7	STA \$F7	
D447	A5 85	LDA \$85	data byte
D449	8D 43 02	STA \$0243	into output register
D44C	60	RTS	
*****			read next block
D44D	20 93 DF	JSR \$DF93	get buffer number
D450	0A	ASL A	times 2

D451	AA		TAX	
D452	A9 00		LDA #S00	
D454	95 99		STA \$99,X	buffer pointer to zero
D456	A1 99		LDA (\$99,X)	get first byte from buffer
D458	F0 05		BEQ \$D45F	no block following?
D45A	D6 99		DEC \$99,X	buffer pointer to -1
D45C	4C 56 D1		JMP \$D156	read next block
D45F	60		RTS	
*****				read block
D460	A9 80		LDA #S80	command code for reading
D462	D0 02		BNE \$D466	
*****				write block
D464	A9 90		LDA #S90	command code for writing
D466	05 7F		ORA \$7F	drive number
D468	8D 4D 02		STA \$024D	save code
D46B	A5 F9		LDA \$F9	
D46D	20 D3 D6		JSR \$D6D3	param to disk controller
D470	A6 F9		LDX \$F9	
D472	4C 93 D5		JMP \$D593	execute command
*****				allocate buffer and read block
D475	A9 01		LDA #S01	
D477	8D 4A 02		STA \$024A	file type to sequential
D47A	A9 11		LDA #S11	17
D47C	85 83		STA \$83	secondary address
D47E	20 46 DC		JSR \$DC46	allocate buffer and read block
D481	A9 02		LDA #S02	
D483	4C C8 D4		JMP \$D4C8	buffer pointer to 2
*****				allocate new block
D486	A9 12		LDA #S12	18
D488	85 83		STA \$83	secondary address
D48A	4C DA DC		JMP \$DCDA	allocate new block
*****				write directory block
D48D	20 3B DE		JSR \$DE3B	get track and sector number
D490	A9 01		LDA #S01	
D492	85 6F		STA \$6F	a block
D494	A5 69		LDA \$69	save step width 10 for block
D496	48		PHA	allocation
D497	A9 03		LDA #S03	
D499	85 69		STA \$69	
D49B	20 2D F1		JSR \$F12D	find free block in BAM
D49E	68		PLA	
D49F	85 69		STA \$69	get step width back
D4A1	A9 00		LDA #S00	
D4A3	20 C8 D4		JSR \$D4C8	buffer pointer to zero
D4A6	A5 80		LDA \$80	
D4A8	20 F1 CF		JSR \$CFF1	track number in buffer
D4AB	A5 81		LDA \$81	
D4AD	20 F1 CF		JSR \$CFF1	sector number in buffer
D4B0	20 C7 D0		JSR \$D0C7	write block to disk
D4B3	20 99 D5		JSR \$D599	and verify

# Anatomy of the 1541 Disk Drive

D4B6	A9 00	LDA #\$00	
D4B8	20 C8 D4	JSR \$D4C8	buffer pointer to zero
D4BB	20 F1 CF	JSR \$CFF1	fill buffer with zeroes
D4BE	D0 FB	BNE \$D4BB	
D4C0	20 F1 CF	JSR \$CFF1	zero as following track
D4C3	A9 FF	LDA #\$FF	
D4C5	4C F1 CF	JMP \$CFF1	\$FF as number of bytes
*****			
D4C8	85 6F	STA \$6F	set buffer pointer
D4CA	20 93 DF	JSR \$DF93	save pointer
D4CD	0A	ASL A	get buffer number
D4CE	AA	TAX	times 2
D4CF	B5 9A	LDA \$9A,X	buffer pointer hi
D4D1	85 95	STA \$95	
D4D3	A5 6F	LDA \$6F	
D4D5	95 99	STA \$99,X	buffer pointer lo, new value
D4D7	85 94	STA \$94	
D4D9	60	RTS	
*****			
D4DA	A9 11	LDA #\$11	close internal channel
D4DC	85 83	STA \$83	17
D4DE	20 27 D2	JSR \$D227	close channel
D4E1	A9 12	LDA #\$12	18
D4E3	85 83	STA \$83	
D4E5	4C 27 D2	JMP \$D227	close channel
*****			
D4E8	20 93 DF	JSR \$DF93	set buffer pointer
D4EB	0A	ASL A	get buffer number
D4EC	AA	TAX	
D4ED	B5 9A	LDA \$9A,X	buffer pointer hi
D4EF	85 95	STA \$95	
D4F1	B5 99	LDA \$99,X	buffer pointer lo
D4F3	85 94	STA \$94	
D4F5	60	RTS	
*****			
D4F6	85 71	STA \$71	get byte from buffer
D4F8	20 93 DF	JSR \$DF93	pointer lo
D4FB	AA	TAX	get buffer number
D4FC	BD E0 FE	LDA \$FEE0,X	hi-byte buffer address
D4FF	85 72	STA \$72	pointer hi
D501	A0 00	LDY #\$00	
D503	B1 71	LDA (\$71),Y	get byte from buffer
D505	60	RTS	
*****			
D506	BD 5B 02	LDA \$025B,X	check track and sector numbers
D509	29 01	AND #\$01	command code for disk controller
D50B	0D 4D 02	ORA \$024D	drive number
D50E	A8	PHA	plus command code
D50F	86 F9	STX \$F9	save
D511	8A	TXA	buffer number

# Anatomy of the 1541 Disk Drive

D512	0A	ASL A	times 2
D513	AA	TAX	
D514	B5 07	LDA \$07,X	sector
D516	8D 4D 02	STA \$024D	save
D519	B5 06	LDA \$06,X	track
D51B	F0 2D	BEQ \$D54A	66, 'illegal track or sector'
D51D	CD D7 FE	CMP \$FED7	36, highest track number + 1
D520	B0 28	BCS \$D54A	66, 'illegal track or sector'
D522	AA	TAX	
D523	68	PLA	command code
D524	48	PHA	
D525	29 F0	AND #\$F0	
D527	C9 90	CMP #\$90	code for writing?
D529	D0 4F	BNE \$D57A	no
D52B	68	PLA	
D52C	48	PHA	
D52D	4A	LSR A	
D52E	B0 05	BCS \$D535	
D530	AD 01 01	LDA \$0101	
D533	90 03	BCC \$D538	
D535	AD 02 01	LDA \$0102	
D538	F0 05	BEQ \$D53F	
D53A	CD D5 FE	CMP \$FED5	'A', format marker
D53D	D0 33	BNE \$D572	73, 'cbm dos v2.6 1541'
D53F	8A	TXA	track number
D540	20 4B F2	JSR \$F24B	get maximum sector number
D543	CD 4D 02	CMP \$024D	compare with sector number
D546	F0 02	BEQ \$D54A	equal, then error
D548	B0 30	BCS \$D57A	smaller?
D54A	20 52 D5	JSR \$D552	get track and sector number
D54D	A9 66	LDA #\$66	
D54F	4C 45 E6	JMP \$E645	66, 'illegal track or sector'
*****			get track and sector number
D552	A5 F9	LDA \$F9	buffer number
D554	0A	ASL A	*2
D555	AA	TAX	as index
D556	B5 06	LDA \$06,X	
D558	85 80	STA \$80	track
D55A	B5 07	LDA \$07,X	
D55C	85 81	STA \$81	sector
D55E	60	RTS	
D55F	A5 80	LDA \$80	track
D561	F0 EA	BEQ \$D54D	zero, then error
D563	CD D7 FE	CMP \$FED7	36, maximum track number + 1
D566	B0 E5	BCS \$D54D	66, 'illegal track or sector'
D568	20 4B F2	JSR \$F24B	get maximum sector number
D56B	C5 81	CMP \$81	sector
D56D	F0 DE	BEQ \$D54D	
D56F	90 DC	BCC \$D54D	error
D571	60	RTS	
D572	20 52 D5	JSR \$D552	get track and sector number
D575	A9 73	LDA #\$73	



# Anatomy of the 1541 Disk Drive

D577	4C 45 E6	JMP \$E645	73, 'cbm dos v2.6 1541'
D57A	A6 F9	LDX \$F9	buffer number
D57C	68	PLA	
D57D	8D 4D 02	STA \$024D	command code for disk controller
D580	95 00	STA \$00,X	in command register
D582	9D 5B 02	STA \$025B,X	and write in table
D585	60	RTS	
***** read block			
D586	A9 80	LDA #\$80	code for read
D588	D0 02	BNE \$D58C	
***** write block			
D58A	A9 90	LDA #\$90	code for write
D58C	05 7F	ORA \$7F	drive number
D58E	A6 F9	LDX \$F9	buffer number
D590	8D 4D 02	STA \$024D	
D593	AD 4D 02	LDA \$024D	command code
D596	20 0E D5	JSR \$D50E	check track and sector
***** verify execution			
D599	20 A6 D5	JSR \$D5A6	verify execution
D59C	B0 FB	BCS \$D599	wait for end
D59E	48	PHA	
D59F	A9 00	LDA #\$00	
D5A1	8D 98 02	STA \$0298	erase error flag
D5A4	68	PLA	
D5A5	60	RTS	
D5A6	F5 00	LDA \$00,X	cmd code (bit 7) still in reg?
D5A8	30 1A	BMI \$D5C4	yes
D5AA	C9 02	CMP #\$02	
D5AC	90 14	BCC \$D5C2	error-free execution
D5AE	C9 08	CMP #\$08	8
D5B0	F0 08	BEQ \$D5BA	write protect
D5B2	C9 0B	CMP #\$0B	11
D5B4	F0 04	BEQ \$D5BA	ID mismatch
D5B6	C9 0F	CMP #\$0F	15
D5B8	D0 0C	BNE \$D5C6	
D5BA	2C 98 02	BIT \$0298	
D5BD	30 03	BMI \$D5C2	
D5BF	4C 3F D6	JMP \$D63F	create error message
D5C2	18	CLC	execution ended
D5C3	60	RTS	
D5C4	38	SEC	execution not yet ended
D5C5	60	RTS	
D5C6	98	TYA	
D5C7	48	PHA	
D5C8	A5 7F	LDA \$7F	drive number
D5CA	48	PHA	
D5CB	BD 5B 02	LDA \$025B,X	

# Anatomy of the 1541 Disk Drive

D5CE	29	01	AND #S01	drive number
D5D0	85	7F	STA \$7F	
D5D2	A8		TAY	
D5D3	B9	CA FE	LDA \$FECA,Y	bit model for drive
D5D6	8D	6D 02	STA \$026D	
D5D9	20	A6 D6	JSR \$D6A6	read attempt
D5DC	C9	02	CMP #S02	
D5DE	B0	03	BCS \$D5E3	not ok?
D5E0	4C	6D D6	JMP \$D66D	done
D5E3	BD	5B 02	LDA \$025B,X	command code
D5E6	29	F0	AND #\$F0	isolate
D5E8	48		PHA	
D5E9	C9	90	CMP #\$90	code for write
D5EB	D0	07	BNE \$D5F4	no
D5ED	A5	7F	LDA \$7F	drive number
D5EF	09	B8	ORA #\$B8	
D5F1	9D	5B 02	STA \$025B,X	
D5F4	24	6A	BIT \$6A	
D5F6	70	39	BVS \$D631	
D5F8	A9	00	LDA #\$00	
D5FA	8D	99 02	STA \$0299	cntr for searches next to track
D5FD	8D	9A 02	STA \$029A	
D600	AC	99 02	LDY \$0299	counter
D603	AD	9A 02	LDA \$029A	
D606	38		SEC	
D607	F9	DB FE	SBC \$FEDB,Y	constants for read attempts
D60A	8D	9A 02	STA \$029A	
D60D	B9	DB FE	LDA \$FEDB,Y	
D610	20	76 D6	JSR \$D676	position head next to track
D613	EE	99 02	INC \$0299	increment counter
D616	20	A6 D6	JSR \$D6A6	read attempt
D619	C9	02	CMP #S02	return message
D61B	90	08	BCC \$D625	smaller than 2, ok?
D61D	AC	99 02	LDY \$0299	load counter
D620	B9	DB FE	LDA \$FEDB,Y	get constants
D623	D0	DB	BNE \$D600	not yet zero (table end)?
D625	AD	9A 02	LDA \$029A	
D628	20	76 D6	JSR \$D676	position head
D62B	B5	00	LDA \$00,X	
D62D	C9	02	CMP #S02	return message
D62F	90	2B	BCC \$D65C	ok?
D631	24	6A	BIT \$6A	
D633	10	0F	BPL \$D644	
D635	68		PLA	command code
D636	C9	90	CMP #\$90	for writing?
D638	D0	05	BNE \$D63F	no
D63A	05	7F	ORA \$7F	drive number
D63C	9D	5B 02	STA \$025B,X	command code in table
D63F	B5	00	LDA \$00,X	return message
D641	20	0A E6	JSR \$E60A	set error message
D644	68		PLA	
D645	2C	98 02	BIT \$0298	
D648	30	23	BMI \$D66D	
D64A	48		PHA	
D64B	A9	C0	LDA #\$C0	command code for head positioning

# Anatomy of the 1541 Disk Drive

D64D	05 7F	ORA \$7F	drive number
D64F	95 00	STA \$00,X	in command register
D651	B5 00	LDA \$00,X	
D653	30 FC	BMI \$D651	wait for execution
D655	20 A6 D6	JSR \$D6A6	attempt command execution again
D658	C9 02	CMP #\$02	return message
D65A	B0 D9	BCS \$D635	incorrect?
D65C	68	PLA	
D65D	C9 90	CMP #\$90	command code for writing
D65F	D0 0C	BNE \$D66D	no
D661	05 7F	ORA \$7F	drive number
D663	9D 5B 02	STA \$025B,X	in table
D666	20 A6 D6	JSR \$D6A6	attempt execution again
D669	C9 02	CMP #\$02	return message
D66B	B0 D2	BCS \$D63F	error?
D66D	68	PLA	
D66E	85 7F	STA \$7F	get drive number back
D670	68	PLA	
D671	A8	TAY	
D672	B5 00	LDA \$00,X	error code
D674	18	CLC	end-of-execution flag
D675	60	RTS	
D676	C9 00	CMP #\$00	
D678	F0 18	BEQ \$D692	
D67A	30 0C	BMI \$D688	
D67C	A0 01	LDY #\$01	
D67E	20 93 D6	JSR \$D693	transmit data for head position
D681	38	SEC	
D682	E9 01	SBC #\$01	
D684	D0 F6	BNE \$D67C	
D686	F0 0A	BEQ \$D692	
D688	A0 FF	LDY #\$FF	
D68A	20 93 D6	JSR \$D693	transmit data for head position
D68D	18	CLC	
D68E	69 01	ADC #\$01	
D690	D0 F6	BNE \$D688	
D692	60	RTS	
D693	48	PHA	
D694	98	TYA	
D695	A4 7F	LDY \$7F	drive number
D697	99 FE 02	STA \$02FE,Y	
D69A	D9 FE 02	CMP \$02FE,Y	wait for return message from
D69D	F0 FB	BEQ \$D69A	
D69F	A9 00	LDA #\$00	disk controller
D6A1	99 FE 02	STA \$02FE,Y	
D6A4	68	PLA	
D6A5	60	RTS	
D6A6	A5 6A	LDA \$6A	maximum number of repetitions
D6A8	29 3F	AND #\$3F	
D6AA	A8	TAY	
D6AB	AD 6D 02	LDA \$026D	bit for LED

# Anatomy of the 1541 Disk Drive

D6AE	4D 00 1C	EOR \$1C00	
D6B1	8D 00 1C	STA \$1C00	
D6B4	BD 5B 02	LDA \$025B,X	command
D6B7	95 00	STA \$00,X	transmit to disk controller
D6B9	B5 00	LDA \$00,X	and return message
D6BB	30 FC	BMI \$D6B9	wait
D6BD	C9 02	CMP #\$02	ok?
D6BF	90 03	BCC \$D6C4	yes
D6C1	88	DEY	decrement counter
D6C2	D0 E7	BNE \$D6AB	attempt again
D6C4	48	PHA	
D6C5	AD 6D 02	LDA \$026D	
D6C8	0D 00 1C	ORA \$1C00	LED off
D6CB	8D 00 1C	STA \$1C00	
D6CE	68	PLA	
D6CF	60	RTS	
*****			transmit param to disk controller
D6D0	20 93 DF	JSR \$DF93	get buffer number
D6D3	0A	ASL A	
D6D4	A8	TAY	
D6D5	A5 80	LDA \$80	track number
D6D7	99 06 00	STA \$0006,Y	transmit
D6DA	A5 81	LDA \$81	sector number
D6DC	99 07 00	STA \$0007,Y	transmit
D6DF	A5 7F	LDA \$7F	drive number
D6E1	0A	ASL	times 2
D6E2	AA	TAX	
D6E3	60	RTS	
*****			enter file in directory
D6E4	A5 83	LDA \$83	secondary address
D6E6	48	PHA	
D6E7	A5 82	LDA \$82	channel number
D6E9	48	PHA	
D6EA	A5 81	LDA \$81	sector number
D6EC	48	PHA	
D6ED	A5 80	LDA \$80	track number
D6EF	48	PHA	save
D6F0	A9 11	LDA #\$11	
D6F2	85 83	STA \$83	secondary address 17
D6F4	20 3B DE	JSR \$DE3B	get track and sector number
D6F7	AD 4A 02	LDA \$024A	file type
D6FA	48	PHA	save
D6FB	A4 E2	LDA \$E2	drive number
D6FD	29 01	AND #\$01	
D6FF	85 7F	STA \$7F	set
D701	A6 F9	LDX \$F9	buffer number
D703	5D 5B 02	EOR \$025B,X	
D706	4A	LSR A	
D707	90 0C	BCC \$D715	equal drive number?
D709	A2 01	LDX #\$01	
D70B	8E 92 02	STX \$0292	pointer in directory
D70E	20 AC C5	JSR \$C5AC	load dir and find first entry
D711	F0 1D	BEQ \$D730	not found?

# Anatomy of the 1541 Disk Drive

D713	D0 28	BNE \$D73D	found?
D715	AD 91 02	LDA \$0291	sector number in directory
D718	F0 0C	BEQ \$D726	equal zero
D71A	C5 81	CMP \$81	equal sector number?
D71C	F0 1F	BEQ \$D73D	yes
D71E	85 81	STA \$81	save sector number
D720	20 60 D4	JSR \$D460	read block
D723	4C 3D D7	JMP \$D73D	
D726	A9 01	LDA #\$01	
D728	8D 92 02	STA \$0292	pointer to one
D72B	20 17 C6	JSR \$C617	find next entry in directory
D72E	D0 0D	BNE \$D73D	found?
D730	20 8D D4	JSR \$D48D	write directory block
D733	A5 81	LDA \$81	sector number
D735	8D 91 02	STA \$0291	
D738	A9 02	LDA #\$02	
D73A	8D 92 02	STA \$0292	pointer to 2
D73D	AD 92 02	LDA \$0292	
D740	20 C8 D4	JSR \$D4C8	set buffer pointer
D743	68	PLA	
D744	8D 4A 02	STA \$024A	file type
D747	C9 04	CMP #\$04	rel-file?
D749	D0 02	BNE \$D74D	no
D74B	09 80	ORA #\$80	set bit 7
D74D	20 F1 CF	JSR \$CFF1	and write in buffer
D750	68	PLA	
D751	8D 80 02	STA \$0280	following track
D754	20 F1 CF	JSR \$CFF1	in buffer
D757	68	PLA	
D758	8D 85 02	STA \$0285	following sector
D75B	20 F1 CF	JSR \$CFF1	in buffer
D75E	20 93 DF	JSR \$DF93	get buffer number
D761	A8	TAY	
D762	AD 7A 02	LDA \$027A	pointer to drive number
D765	AA	TAX	
D766	A9 10	LDA #\$10	16, length of filename
D768	20 6E C6	JSR \$C66E	write filename in buffer
D76B	A0 10	LDY #\$10	
D76D	A9 00	LDA #\$00	
D76F	91 94	STA (\$94),Y	fill with zeroes at pos 16
D771	C8	INY	
D772	C0 1B	CPY #\$1B	position 27 already?
D774	90 F9	BCC \$D76F	no
D776	AD 4A 02	LDA \$024A	file type
D779	C9 04	CMP #\$04	rel-file
D77B	D0 13	BNE \$D790	no
D77D	A0 10	LDY #\$10	
D77F	AD 59 02	LDA \$0259	track
D782	91 94	STA (\$94),Y	
D784	C8	INY	
D785	AD 5A 02	LDA \$025A	and sector
D788	91 94	STA (\$94),Y	the side-sectors in dir entry
D78A	C8	INY	

# Anatomy of the 1541 Disk Drive

D78B	AD 58 02	LDA \$0258	record length
D78E	91 94	STA (\$94),Y	in directory
D790	20 64 D4	JSR \$D464	write block
D793	68	PLA	
D794	85 82	STA \$82	channel number
D796	AA	TAX	
D797	68	PLA	
D798	85 83	STA \$83	secondary address
D79A	AD 91 02	LDA \$0291	
D79D	85 D8	STA \$D8	
D79F	9D 60 02	STA \$0260,X	
D7A2	AD 92 02	LDA \$0292	
D7A5	85 DD	STA \$DD	
D7A7	9D 66 02	STA \$0266,X	
D7AA	AD 4A 02	LDA \$024A	file type
D7AD	85 E7	STA \$E7	
D7AF	A5 7F	LDA \$7F	drive number
D7B1	85 E2	STA \$E2	
D7B3	60	RTS	
*****			
D7B4	A5 83	LDA \$83	OPEN command, secondary adr <> 15
D7B6	8D 4C 02	STA \$024C	secondary address
D7B9	20 B3 C2	JSR \$C283	get line length, erase flags
D7BC	8E 2A 02	STX \$022A	
D7BF	AE 00 02	LDX \$0200	first character from buffer
D7C2	AD 4C 02	LDA \$024C	secondary address
D7C5	D0 2C	BNE \$D7F3	not equal 0 (LOAD)?
D7C7	E0 2A	CPX #\$2A	'**'
D7C9	D0 28	RNE \$D7F3	
D7CB	A5 7E	LDA \$7E	last track number
D7CD	F0 4D	BEQ \$D81C	
D7CF	85 80	STA \$80	track number
D7D1	AD 6E 02	LDA \$026E	last drive number
D7D4	85 7F	STA \$7F	drive number
D7D6	85 E2	STA \$E2	
D7D8	A9 02	LDA #\$02	
D7DA	85 E7	STA \$E7	set data type to program
D7DC	AD 6F 02	LDA \$026F	last sector number
D7DF	85 81	STA \$81	sector
D7E1	20 00 C1	JSR \$C100	turn LED on
D7E4	20 46 DC	JSR \$DC46	allocate buffer, read block
D7E7	A9 04	LDA #\$04	file type
D7E9	05 7F	ORA \$7F	drive number
D7EB	A6 82	LDX \$82	channel number
D7ED	99 EC 00	STA \$00EC,Y	set flag
D7F0	4C 94 C1	JMP \$C194	done
D7F3	E0 24	CPX #\$24	'\$'
D7F5	D0 1E	BNE \$D815	no
D7F7	AD 4C 02	LDA \$024C	secondary address
D7FA	D0 03	BNE \$D7FF	not equal to zero?
D7FC	4C 55 DA	JMP \$DA55	OPEN \$
D7FF	20 D1 C1	JSR \$C1D1	analyze line to end

# Anatomy of the 1541 Disk Drive

D802	AD 85 FE	LDA \$FE85	18, directory track
D805	85 80	STA \$80	track
D807	A9 00	LDA #\$00	
D809	85 81	STA \$81	sector 0
D80B	20 46 DC	JSR \$DC46	allocate buffer, read block
D80E	A5 7F	LDA \$7F	drive number
D810	09 02	ORA #\$02	
D812	4C EB D7	JMP \$D7EB	continue as above
D815	E0 23	CPX #\$23	'#'
D817	D0 12	RNE \$D82B	
D819	4C 84 CB	JMP \$CB84	open direct access file
D81C	A9 02	LDA #\$02	
D81E	8D 96 02	STA \$0296	file type program
D821	A9 00	LDA #\$00	
D823	85 7F	STA \$7F	drive 0
D825	8D 8E 02	STA \$028E	
D828	20 42 D0	JSR \$D042	load BAM
D82B	20 E5 C1	JSR \$C1E5	analyze line
D82E	D0 04	BNE \$D834	colon found?
D830	A2 00	LDX #\$00	
D832	F0 0C	BEQ \$D840	
D834	8A	TXA	comma found?
D835	F0 05	REQ \$D83C	no
D837	A9 30	LDA #\$30	
D839	4C C8 C1	JMP \$C1C8	30, 'syntax error'
D83C	88	DEY	
D83D	F0 01	BEQ \$D840	
D83F	88	DEY	
D840	8C 7A 02	STY \$027A	pointer to drive number
D843	A9 8D	LDA #\$8D	shift CR
D845	20 68 C2	JSR \$C268	analyze line to end
D848	E8	INX	
D849	8E 78 02	STX \$0278	comma counter
D84C	20 12 C3	JSR \$C312	get drive number
D84F	20 CA C3	JSR \$C3CA	check drive number
D852	20 9D C4	JSR \$C49D	find file entry in directory
D855	A2 00	LDX #\$00	default values
D857	8E 58 02	STX \$0258	record length
D85A	8E 97 02	STX \$0297	
D85D	8E 4A 02	STX \$024A	file type
D860	E8	INX	
D861	EC 77 02	CPX \$0277	comma before equal sign?
D864	B0 10	BCS \$D876	no
D866	20 09 DA	JSR \$DA09	get file type and control mode
D869	E8	INX	
D86A	EC 77 02	CPX \$0277	additional comma?
D86D	B0 07	BCS \$D876	no
D86F	C0 04	CPY #\$04	
D871	F0 3E	REQ \$D8B1	
D873	20 09 DA	JSR \$DA09	get file type and control method
D876	AE 4C 02	LDX \$024C	
D879	86 83	STX \$83	secondary address

D87B	E0 02	CPX #\$02	greater than 2?
D87D	B0 12	BCS \$D891	yes
D87F	8E 97 02	STX \$0297	0 or 1 (LOAD or SAVE)
D882	A9 40	LDA #\$40	
D884	8D F9 02	STA \$02F9	
D887	AD 4A 02	LDA \$024A	file type
D88A	D0 1B	BNE \$D8A7	not deleted
D88C	A9 02	LDA #\$02	PRG
D88E	8D 4A 02	STA \$024A	as file type
D891	AD 4A 02	LDA \$024A	
D894	D0 11	BNE \$D8A7	
D896	A5 E7	LDA \$E7	
D898	29 07	AND #\$07	get file type and command line
D89A	8D 4A 02	STA \$024A	
D89D	AD 80 02	LDA \$0280	track number
D8A0	D0 05	BNE \$D8A7	not equal zero?
D8A2	A9 01	LDA #\$01	
D8A4	8D 4A 02	STA \$024A	file type sequential
D8A7	AD 97 02	LDA \$0297	control method
DBAA	C9 01	CMP #\$01	'W'
DBAC	F0 18	BEQ \$D8C6	yes
DBAE	4C 40 D9	JMP \$D940	
D8B1	BC 7A 02	LDY \$027A,X	pointer behind second comma
D8B4	B9 00 02	LDA \$0200,Y	get value
D8B7	8D 5B 02	STA \$025B	record length
D8BA	AD 80 02	LDA \$0280	track number
D8BD	D0 B7	BNE \$D876	
D8BF	A9 01	LDA #\$01	'W'
D8C1	8D 97 02	STA \$0297	as control method
D8C4	D0 B0	BNE \$D876	
D8C6	A5 E7	LDA \$E7	file type
D8C8	29 80	AND #\$80	isolate wildcard flag
D8CA	AA	TAX	
D8CB	D0 14	BNE \$D8E1	wildcard in name
D8CD	A9 20	LDA #\$20	
D8CF	24 E7	BIT \$E7	was file closed?
D8D1	F0 06	BEQ \$D8D9	yes
D8D3	20 B6 C8	JSR \$C8B6	byte 0 in buffer and write block
D8D6	4C E3 D9	JMP \$D9E3	
D8D9	A9 80 02	LDA \$0280	track number of the first block
D8DC	D0 03	BNE \$D8E1	already existing
D8DE	4C E3 D9	JMP \$D9E3	
D8E1	AD 00 02	LDA \$0200	first character from input buffer
D8E4	C9 40	CMP #\$40	'@'?
D8E6	F0 0D	BEQ \$D8F5	yes
D8E8	8A	TXA	
D8E9	D0 05	BNE \$D8F0	wildcard set?
D8EB	A9 63	LDA #\$63	
D8ED	4C C8 C1	JMP \$C1C8	63, 'file exists'
D8F0	A9 33	LDA #\$33	
D8F2	4C C8 C1	JMP \$C1C8	33, 'syntax error'



# Anatomy of the 1541 Disk Drive

```

*****
D8F5  A5 E7      LDA $E7      open a file with overwriting
D8F7  29 07      AND #$07     file type
D8F9  CD 4A 02   CMP $024A    isolate
D8FC  D0 67      BNE $D965    file type different?
D8FE  C9 04      CMP #$04     rel-file?
D900  F0 63      BEQ $D965    64, 'file type mismatch'
D902  20 DA DC   JSR $DCDA
D905  A5 82      LDA $82
D907  8D 70 02   STA $0270    save channel number
D90A  A9 11      LDA #$11
D90C  20 EB D0   JSR $D0EB    open read channel
D911  AD 94 02   LDA $0294
D914  20 C8 D4   JSR $D4C8    set buffer pointer for directory
D917  A0 00      LDY #$00
D919  B1 94      LDA ($94),Y  file type
D91B  09 20      ORA #$20     set bit 5, open file
D91D  91 94      STA ($94),Y
D91F  A0 1A      LDY #$1A
D921  A5 80      LDA $80     track
D923  91 94      STA ($94),Y
D925  C8         INY
D926  A5 81      LDA $81     and sector
D928  91 94      STA ($94),Y  for open with at-sign
D92A  AE 70 02   LDX $0270    channel number
D92D  A5 D8      LDA $D8
D92F  9D 60 02   STA $0260,X  pointer to directory block
D932  A5 DD      LDA $DD
D934  9D 66 02   STA $0266,X
D937  20 3B DE   JSR $DE3B    get track and sector number
D93A  20 64 D4   JSR $D464    write block
D93D  4C EF D9   JMP $D9EF    prepare trk, sector, and drive #

D940  AD 80 02   LDA $0280    first track number
D943  D0 05      BNE $D94A    file not erased?
D945  A9 62      LDA #$62
D947  4C C8 C1   JMP $C1C8    62, 'file not found'
D94A  AD 97 02   LDA $0297    control mode
D94D  C9 03      CMP #$03     'M'
D94F  F0 0B      BEQ $D95C    yes, then no test of unclosed file
D951  A9 20      LDA #$20     bit 5
D953  24 E7      BIT $E7     test in file type
D955  F0 05      BEQ $D95C    not set, ok
D957  A9 60      LDA #$60
D959  4C C8 C1   JMP $C1C8    60, 'write file open'
D95C  A5 F7      LDA $E7
D95E  29 07      AND #$07     isolate file type
D960  CD 4A 02   CMP $024A
D963  F0 05      BEQ $D96A
D965  A9 64      LDA #$64
D967  4C C8 C1   JMP $C1C8    64, 'file type mismatch'
D96A  A0 00      LDY #$00
D96C  8C 79 02   STY $0279
D96F  AE 97 02   LDX $0297    control mode
D972  E0 02      CPX #$02     'A', append

```

# Anatomy of the 1541 Disk Drive

D974	D0 1A	BNE \$D990	no
D976	C9 04	CMP #\$04	rel-file?
D978	F0 EB	BEO \$D965	
D97A	B1 94	LDA (\$94),Y	
D97C	29 4F	AND #\$4F	
D97E	91 94	STA (\$94),Y	
D980	A5 83	LDA \$83	
D982	48	PHA	
D983	A9 11	LDA #\$11	
D985	85 83	STA \$83	channel 17
D987	20 3B DE	JSR \$DE3B	get track and sector number
D98A	20 64 D4	JSR \$D464	write block
D98D	68	PLA	
D98E	85 83	STA \$83	get channel # back
D990	20 A0 D9	JSR \$D9A0	
D993	AD 97 02	LDA \$0297	control mode
D996	C9 02	CMP #\$02	
D998	D0 55	BNE \$D9EF	
D99A	20 2A DA	JSR \$DA2A	
D99D	4C 94 C1	JMP \$C194	done
D9A0	A0 13	LDA #\$13	
D9A2	B1 94	LDA (\$94),Y	track
D9A4	8D 59 02	STA \$0259	
D9A7	C8	INY	
D9A8	B1 94	LDA (\$94),Y	
D9AA	8D 5A 02	STA \$025A	
D9AD	C8	INY	
D9AE	B1 94	LDA (\$94),Y	record length
D9B0	AE 58 02	LDX \$0258	last record len
D9B3	8D 58 02	STA \$0258	
D9B6	8A	TXA	
D9B7	F0 0A	BEO \$D9C3	
D9B9	CD 58 02	CMP #\$0258	
D9BC	F0 05	BEO \$D9C3	
D9BE	A9 50	LDA #\$50	
D9C0	20 C8 C1	JSR \$C1C8	50, 'record not present'
D9C3	AE 79 02	LDX \$0279	
D9C6	BD 80 02	LDA \$0280,X	
D9C9	85 80	STA \$80	track
D9CB	BD 85 02	LDA \$0285,X	
D9CE	85 81	STA \$81	sector
D9D0	20 46 DC	JSR \$DC46	
D9D3	A4 82	LDY \$82	
D9D5	AE 79 02	LDX \$0279	
D9D8	B5 D8	LDA \$D8,X	
D9DA	99 60 02	STA \$0260,Y	
D9DD	B5 DD	LDA \$DD,X	
D9DF	99 66 02	STA \$0266,Y	
D9E2	60	RTS	
D9E3	A5 E2	LDA \$E2	drive #
D9E5	29 01	AND #\$01	
D9F7	85 7F	STA \$7F	
D9E9	20 DA DC	JSR \$DCDA	

# Anatomy of the 1541 Disk Drive

D9EC	20 E4 D6	JSR \$D6E4	
D9EF	A5 83	LDA \$83	channel #
D9F1	C9 02	CMP #\$02	
D9F3	B0 11	BCS \$DA06	
D9F5	20 3E DE	JSR \$DE3E	
D9F8	A5 80	LDA \$80	
D9FA	85 7E	STA \$7E	
D9FC	A5 7F	LDA \$7F	
D9FE	8D 6E 02	STA \$026E	
DA01	A5 81	LDA \$81	
DA03	8D 6F 02	STA \$026F	
DA06	4C 99 C1	JMP \$C199	
***** check file type and control mode			
DA09	BC 7A 02	LDY \$027A,X	pointer in command line
DA0C	B9 00 02	LDA \$0200,Y	get characters from line
DA0F	A0 04	LDY #\$04	
DA11	88	DEY	
DA12	30 08	BMI \$DA1C	
DA14	D9 B2 FE	CMP \$FEB2,Y	control modes 'R', 'W', 'A', 'M'
DA17	D0 F8	BNE \$DA11	
DA19	8C 97 02	STY \$0297	save
DA1C	A0 05	LDY #\$05	
DA1E	88	DEY	
DA1F	30 08	BMI \$DA29	
DA21	D9 B6 FE	CMP \$FEB6,Y	file types 'D','S','P','U','L'
DA24	D0 F8	BNE \$DA1E	
DA26	8C 4A 02	STY \$024A	save
DA29	60	RTS	
***** preparation for Append			
DA2A	20 39 CA	JSR \$CA39	open channel to read, get byte
DA2D	A9 80	LDA #\$80	
DA2F	20 A6 DD	JSR \$DDA6	last byte?
DA32	F0 F6	BEQ \$DA 2A	no
DA34	20 95 DE	JSR \$DE95	get track and sector number
DA37	A6 81	LDX \$81	sector number
DA39	E8	INX	
DA3A	8A	TXA	
DA3B	D0 05	BNE \$DA42	not \$FF?
DA3D	20 A3 D1	JSR \$D1A3	close buffer, write block
DA40	A9 02	LDA #\$02	
DA42	20 C8 D4	JSR \$D4C8	buffer pointer to 2
DA45	A6 82	LDX \$82	channel number
DA47	A9 01	LDA #\$01	
DA49	95 F2	STA \$F2,X	set flag for WRITE
DA4B	A9 80	LDA #\$80	
DA4D	05 82	ORA \$82	
DA4F	A6 83	LDX \$83	
DA51	9D 2B 02	STA \$022B,X	channel number in table
DA54	60	RTS	
***** OPEN "\$"			
DA55	A9 0C	LDA #\$0C	command number 12
DA57	8D 2A 02	STA \$022A	

# Anatomy of the 1541 Disk Drive

DA5A	A9 00	LDA #\$00	
DA5C	AE 74 02	LDX \$0274	
DA5F	CA	DEX	
DA60	F0 0B	BEQ \$DA6D	
DA62	CA	DEX	
DA63	D0 21	BNE \$DA86	
DA65	AD 01 02	LDA \$0201	second character
DA68	20 BD C3	JSR \$C3BD	get drive number
DA6B	30 19	BMI \$DA86	not a plain number?
DA6D	85 E2	STA \$E2	
DA6F	EE 77 02	INC \$0277	
DA72	EE 78 02	INC \$0278	
DA75	EE 7A 02	INC \$027A	
DA78	A9 80	LDA #\$80	
DA7A	85 E7	STA \$E7	set wildcard flag
DA7C	A9 2A	LDA #\$2A	'**'
DA7E	8D 00 02	STA \$0200	as file name in command buffer
DA81	8D 01 02	STA \$0201	
DA84	D0 18	BNE \$DA9E	absolute jump
DA86	20 E5 C1	JSR \$C1E5	test input line to ':'
DA89	D0 05	BNE \$DA90	found?
DA8B	20 DC C2	JSR \$C2DC	erase flags
DA8E	A0 03	LDY #\$03	
DA90	88	DEY	
DA91	88	DEY	
DA92	8C 7A 02	STY \$027A	pointer to drive no. in command
DA95	20 00 C2	JSR \$C200	analyze line
DA98	20 98 C3	JSR \$C398	ascertain file type
DA9B	20 20 C3	JSR \$C320	get drive number
DA9E	20 CA C3	JSR \$C3CA	initialize drive if necessary
DAA1	20 B7 C7	JSR \$C7B7	prepare disk title
DAA4	20 9D C4	JSR \$C49D	load directory
DAA7	20 9E EC	JSR \$EC9E	create and prepare directory
DAAA	20 37 D1	JSR \$D137	get byte from buffer
DAAD	A6 82	LDX \$82	channel number
DAAF	9D 3E 02	STA \$023E	byte in output register
DAB2	A4 7F	LDA \$7F	drive number
DAB4	8D 8E 02	STA \$028E	save as last drive number
DAB7	09 04	ORA #\$04	
DAB9	95 EC	STA \$EC,X	PRG-flag
DABB	A9 00	LDA #\$00	
DABD	85 A3	STA \$A3	set pointer back in input buffer
DABF	60	RTS	

*****			CLOSE-routine
DAC0	A9 00	LDA #\$00	
DAC2	8D F9 02	STA \$02F9	
DAC5	A5 83	LDA \$83	secondary address
DAC7	D0 08	BNE \$DAD4	not zero?
DAC9	A9 00	LDA #\$00	secondary address 0, LOAD
DACB	8D 54 02	STA \$0254	
DACE	20 27 D2	JSR \$D227	close channel
DAD1	4C DA D4	JMP \$D4DA	close internal channels 17 & 18
DAD4	C9 0F	CMP #\$0F	15

# Anatomy of the 1541 Disk Drive

DAD6	F0 14	BEQ \$DAEC	yes, close all channels
DAD8	20 02 DB	JSR \$DB02	close file
DADB	A5 83	LDA \$83	secondary address
DADD	C9 02	CMP #\$02	
DADF	90 F0	BCC \$DAD1	smaller than 2?
DAE1	AD 6C 02	LDA \$026C	
DAE4	D0 03	BNE \$DAE9	
DAE6	4C 94 C1	JMP \$C194	termination
DAE9	4C AD C1	JMP \$C1AD	
DAEC	A9 0E	LDA #\$0E	14
DAEE	85 83	STA \$83	secondary address
DAF0	20 02 DB	JSR \$DB02	close file
DAF3	C6 83	DEC \$83	next secondary address
DAF5	10 F9	BPL \$DAF0	
DAF7	AD 6C 02	LDA \$026C	
DAFA	D0 03	BNE \$DAFF	
DAFC	4C 94 C1	JMP \$C194	termination
DAFF	4C AD C1	JMP \$C1AD	
*****			close file
DB02	A6 83	LDX \$83	secondary address
DB04	BD 2B 02	LDA \$022B,X	get channel number
DB07	C9 FF	CMP #\$FF	no channel associated?
DB09	D0 01	BNE \$DB0C	
DB0B	60	RTS	no, then done
DB0C	29 0F	AND #\$0F	isolate channel number
DB0E	85 82	STA \$82	
DB10	20 25 D1	JSR \$D125	check data type
DB13	C9 07	CMP #\$07	direct access?
DB15	F0 0F	BEQ \$DB26	yes
DB17	C9 04	CMP #\$04	rel-file?
DB19	F0 11	BEQ \$DB2C	yes
DB1B	20 07 D1	JSR \$D107	channel for writing open
DB1E	B0 09	BCS \$DB29	no file for writing?
DB20	20 62 DB	JSR \$DB62	write last block
DB23	20 A5 DB	JSR \$DBA5	write entry in dir and block
DB26	20 F4 EE	JSR \$EEF4	write BAM
DB29	4C 27 D2	JMP \$D227	close channel
DR2C	20 F1 DD	JSR \$DDF1	get buffer number, write block
DB2F	20 1E CF	JSR \$CF1E	change buffer
DB32	20 CB E1	JSR \$E1CB	get last side-sector
DB35	A6 D5	LDX \$D5	side-sector number
DB37	86 73	STX \$73	
DB39	E6 73	INC \$73	
DB3B	A9 00	LDA #\$00	
DB3D	85 70	STA \$70	
DR3F	85 71		
DB41	A5 D6	LDA \$D6	
DB43	38	SEC	
DB44	E9 0E	SBC #\$0E	minus 14 for pointer
CB46	85 72	STA \$72	
DB48	20 51 DF	JSR \$DF51	calculate block number of file

# Anatomy of the 1541 Disk Drive

DB4B	A6 82	LDX \$82	channel number
DB4D	A5 70	LDA \$70	
DB4F	95 B5	STA \$B5,X	record number lo
DB51	A5 71	LDA \$71	
DB53	95 BB	STA \$BB,X	record number hi
DB55	A9 40	LDA #\$40	
DB57	20 A6 DD	JSR \$DDA6	bit 6 set?
DB5A	F0 03	BEQ \$DB5F	no
DB5C	20 A5 DB	JSR \$DBA5	enter in directory
DB5F	AC 27 D2	JMP \$D227	close channel

*****			write last block
DB62	A6 82	LDX \$82	channel number
DB64	B5 B5	LDA \$B5,X	record number lo
DB66	15 BB	ORA \$BB,X	record number hi
DB68	D0 0C	BNE \$DB76	not zero?
DB6A	20 E8 D4	JSR \$D4E8	set buffer pointer
DB6D	C9 02	CMP #\$02	
DB6F	D0 05	RNE \$DB76	not 2
DB71	A9 0D	LDA #\$0D	CR
DB73	20 F1 CF	JSR \$CFF1	in buffer
DB76	20 E8 D4	JSR \$D4E8	set buffer pointer
DB79	C9 02	CMP #\$02	now equal to 2?
DB7B	D0 0F	BNE \$DB8C	no
DB7D	20 1E CF	JSR \$CF1E	change buffer
DB80	A6 82	LDX \$82	channel number
DB82	B5 B5	LDA \$B5,X	record number lo
DB84	D0 02	BNE \$DB88	
DB86	D6 BB	DEC \$BB,X	decrement block number hi
DB88	D6 B5	DEC \$B5,X	and block number lo
DB8A	A9 00	LDA #\$00	
DB8C	38	SEC	
DB8D	E9 01	SBC #\$01	set pointer to end
DB8F	48	PHA	
DB90	A9 00	LDA #\$00	
DB92	20 C8 D4	JSR \$D4C8	buffer pointer to zero
DB95	20 F1 CF	JSR \$CFF1	write zero in buffer
DB98	68	PLA	second byte = pointer to end
DB99	20 F1 CF	JSR \$CFF1	write in buffer
DB9C	20 C7 D0	JSR \$D0C7	write block to disk
DB9F	20 99 D5	JSR \$D599	and verify
DBA2	4C 1E CF	JMP \$CF1E	change buffer

*****			directory entry
DBA5	A6 82	LDX \$82	channel number
DBA7	8E 70 02	STX \$0270	save
DBAA	A5 83	LDA \$83	secondary address
DBAC	48	PHA	save
DBAD	BD 60 02	LDA \$0260,X	sector number in directory
DBB0	85 81	STA \$81	set
DBB2	BD 66 02	LDA \$0266,X	pointer in directory
DBB5	8D 94 02	STA \$0294	
DBB8	B5 EC	LDA \$EC,X	
DBBA	29 01	AND #\$01	
DBBC	85 7F	STA \$7F	drive number

# Anatomy of the 1541 Disk Drive

DBBE	AD 85 FE	LDA \$FE85	18, directory track
DBC1	85 80	STA \$80	set
DBC3	20 93 DF	JSR \$DF93	increment buffer number
DBC6	48	PHA	
DBC7	85 F9	STA \$F9	
DBC9	20 60 D4	JSR \$D460	read directory block
DBCC	A0 00	LDY #\$00	
DBCE	BD E0 FE	LDA \$FEE0,X	buffer address
DBD1	85 87	STA \$87	
DBD3	AD 94 02	LDA \$0294	buffer pointer
DBD6	85 86	STA \$86	
DBD8	B1 86	LDA (\$86),Y	file type
DBDA	29 20	AND #\$20	file closed?
DBDC	F0 43	BEQ \$DC21	yes
DBDE	20 25 D1	JSR \$D125	check file type
DBE1	C9 04	CMP #\$04	rel-file?
DBE3	F0 44	BEQ \$DC29	yes
DBE5	B1 86	LDA (\$86),Y	
DBE7	29 8F	AND #\$8F	erase bits 4,5, and 6
DBE9	91 86	STA (\$86),Y	in file type
DBEB	C8	INY	
DBEC	B1 86	LDA (\$86),Y	track number
DBEE	85 80	STA \$80	
DBF0	84 71	STY \$71	
DBF2	A0 1B	LDY #\$1B	
DBF4	B1 86	LDA (\$86),Y	sector # of the file for
DBF6	48	PHA	overwriting
DBF7	88	DEY	
DBF8	B1 86	LDA (\$86),Y	track # for overwriting
DBFA	D0 0A	BNE \$DC06	set?
DBFC	85 80	STA \$80	set track number
DBFE	68	PLA	
DBFF	85 81	STA \$81	sector number
DC01	A9 67	LDA #\$67	
DC03	20 45 E6	JSR \$E645	67, 'illegal track or sector'
DC06	48	PHA	
DC07	A9 00	LDA #\$00	
DC09	91 86	STA (\$86),Y	erase track number
DC0B	C8	INY	
DC0C	91 86	STA (\$86),Y	and sector number of the
DC0E	68	PLA	substitute file
DC0F	A4 71	LDY \$71	
DC11	91 86	STA (\$86),Y	
DC13	C8	INY	set track & sec # of the new file
DC14	B1 86	LDA (\$86),Y	
DC16	85 81	STA \$81	
DC18	68	PLA	
DC19	91 86	STA (\$86),Y	
DC1B	20 7D C8	JSR \$C87D	erase all files
DC1E	4C 29 DC	JMP \$DC29	
DC21	B1 86	LDA (\$86),Y	get file type
DC23	29 0F	AND #\$0F	isolate bits 0-3
DC25	09 80	ORA #\$80	set bit 7 for closed file
DC27	91 86	STA (\$86),Y	

# Anatomy of the 1541 Disk Drive

DC29	AE 70 02	LDX \$0270	channel number
DC2C	A0 1C	LDY #\$1C	
DC2E	B5 B5	LDA \$B5,X	block number lo
DC30	91 86	STA (\$86),Y	in directory entry
DC32	C8	INY	
DC33	B5 BB	LDA \$BB,Y	and block number hi
DC35	91 86	STA (\$86),Y	write
DC37	68	PLA	buffer number
DC38	AA	TAX	
DC39	A9 90	LDA #\$90	code for 'writing'
DC3B	20 90 D5	JSR \$D590	write block
DC40	68	PLA	
DC41	85 83	STA \$83	secondary address
DC43	4C 07 D1	JMP \$D107	open channel for writing
*****			
			read block, layout buffer
DC46	A9 01	LDA #\$01	
DC48	20 E2 D1	JSR \$D1E2	find channel and buffer for read
DC4B	20 B6 DC	JSR \$DCB6	set pointer
DC4E	AD 4A 02	LDA \$024A	file type
DC51	48	PHA	save
DC52	0A	ASL A	
DC53	05 7F	ORA \$7F	drive number
DC55	95 EC	STA \$EC,X	
DC57	20 9B D0	JSR \$D09B	read block in buffer
DC5A	A6 82	LDX \$82	channel number
DC5C	A5 80	LDA \$80	track
DC5E	D0 05	BNE \$DC65	following track?
DC60	A5 81	LDA \$81	sector
DC62	9D 44 02	STA \$0244,X	as end pointer
DC65	68	PLA	file type
DC66	C9 04	CMP #\$04	rel-file?
DC68	D0 3F	BNE \$DCA9	no
DC6A	A4 83	LDA \$83	secondary address
DC6C	B9 2B 02	LDA \$022B,Y	channel number
DC6F	09 40	ORA #\$40	
DC71	99 2B 02	STA \$022B,Y	set flag for READ and WRITE
DC74	AD 58 02	LDA \$0258	record length
DC77	95 C7	STA \$C7,X	
DC79	20 8E D2	JSR \$D28E	find buffer for side-sector
DC7C	10 03	BPL \$DC81	found?
DC7E	4C 0F D2	JMP \$D20F	70, 'no channel'
DC81	A6 82	LDX \$82	channel number
DC83	95 CD	STA \$CD,X	
DC85	AC 59 02	LDY \$0259	
DC88	84 80	STY \$80	track for side-sector
DC8A	AC 5A 02	LDA \$025A	
DC8D	84 81	STY \$81	sector for side-sector
DC8F	20 D3 D6	JSR \$D6D3	transmit parameters to disk cont.
DC92	20 73 DE	JSR \$DE73	read block
DC95	20 99 D5	JSR \$D599	and verify
DC98	A6 82	LDX \$82	channel number
DC9A	A9 02	LDA #\$02	
DC9C	95 C1	STA \$C1,X	pointer for writing



# Anatomy of the 1541 Disk Drive

DC9E	A9 00	LDA #\$00	
DCA0	20 C8 D4	JSR \$D4C8	buffer pointer to zero
DCA3	20 53 E1	JSR \$E153	find next record
DCA6	4C 3E DE	JMP \$DE3E	get track and sector number
DCA9	20 56 D1	JSR \$D156	get byte from buffer
DCAC	A6 82	LDX \$82	channel number
DCAE	9D 3E 02	STA \$023E,X	byte in output register
DCB1	A9 88	LDA #\$88	set flag for READ
DCB3	95 F2	STA \$F2,X	
DCB5	60	RTS	

\*\*\*\*\* reset pointer

DCB6	A6 82	LDX \$82	channel number
DCB8	B5 A7	LDA \$A7,X	buffer number
DCBA	0A	ASL A	times 2
DCBB	A8	TAY	
DCBC	A9 02	LDA #\$02	
DCBE	99 99 00	STA \$0099,Y	buffer pointer lo
DCC1	B5 AE	LDA \$AE,X	
DCC3	09 80	ORA #\$80	set bit 7
DCC5	95 AE	STA \$AE,X	
DCC7	0A	ASL A	
DCC8	A8	TAY	
DCC9	A9 02	LDA #\$02	
DCCB	99 99 00	STA \$0099,Y	buffer pointer lo
DCCF	A9 00	LDA #\$00	
DCD0	95 B5	STA \$B5,X	block number lo
DCD2	95 BB	STA \$BB,X	block number hi
DCD4	A9 00	LDA #\$00	
DCD6	9D 44 02	STA \$0244,X	end pointer
DCD9	60	RTS	

\*\*\*\*\* construct a new block

DCDA	20 A9 F1	JSR \$F1A9	find free sector in BAM
DCDD	A9 01	LDA #\$01	
DCDF	20 DF D1	JSR \$D1DF	open channel
DCE2	20 D0 D6	JSR \$D6D0	transmit param to disk controller
DCE5	20 B6 DC	JSR \$DCB6	reset pointer
DCE8	A6 82	LDX \$82	channel number
DCEA	AD 4A 02	LDA \$024A	file type
DCED	48	PHA	
DCEE	0A	ASL A	
DCEF	05 7F	ORA \$7F	drive number
DCF1	95 EC	STA \$EC,X	save as flag
DCF3	68	PLA	
DCF4	C9 04	CMP #\$04	rel-file?
DCF6	F0 05	BEQ \$DCFD	yes
DCF8	A9 01	LDA #\$01	
DCFA	95 F2	STA \$F2,X	set WRITE flag
DCFC	60	RTS	
DCFD	A4 83	LDY \$83	secondary address
DCFF	B9 2B 02	LDA \$022B,Y	channel number in table
DD02	29 3F	AND #\$3F	erase the top two bits

# Anatomy of the 1541 Disk Drive

DD04	09 40	ORA #\$40	set bit 6
DD06	99 2B 02	STA \$022B,Y	READ and WRITE flag
DD09	AD 58 02	LDA \$0258	record length
DD0C	95 C7	STA \$C7,X	in table
DD0E	20 8E D2	JSR \$D28E	find buffer
DD11	10 03	BPL \$DD16	found?
DD13	4C 0F D2	JMP \$D20F	70, 'no channel'
DD16	A6 82	LDX \$82	channel number
DD18	95 CD	STA \$CD,X	buffer number for side-sector
DD1A	20 C1 DE	JSR \$DEC1	erase buffer
DD1D	20 1E F1	JSR \$F11E	find free block in BAM
DD20	A5 80	LDA \$80	track
DD22	8D 59 02	STA \$0259	for side-sector
DD25	A5 81	LDA \$81	sector
DD27	8D 5A 02	STA \$025A	for side-sector
DD2A	A6 82	LDX \$82	channel number
DD2C	B5 CD	LDA \$CD,X	buffer number
DD2E	20 D3 D6	JSR \$D6D3	transmit param to disk controller
DD31	A9 00	LDA #\$00	
DD33	20 E9 DE	JSR \$DEE9	buffer pointer to zero
DD36	A9 00	LDA #\$00	
DD38	20 8D DD	JSR \$DD8D	
DD3B	A9 11	LDA #\$11	17
DD3D	20 8D DD	JSR \$DD8D	as end pointer in buffer
DD40	A9 00	LDA #\$00	zero
DD42	20 8D DD	JSR \$DD8D	as side-sector number in buffer
DD45	AD 58 02	LDA \$0258	record length
DD48	20 8D DD	JSR \$DD8D	in buffer
DD4B	A5 80	LDA \$80	track number of this block
DD4D	20 8D DD	JSR \$DD8D	in buffer
DD50	A5 81	LDA \$81	sector number
DD52	20 8D DD	JSR \$DD8D	in buffer
DD55	A9 10	LDA #\$10	16
DD57	20 E9 DE	JSR \$DEE9	buffer pointer to 16
DD5A	20 3E DE	JSR \$DE3E	get track and sector number
DD5D	A5 80	LDA \$80	track # of the first data block
DD5F	20 8D DD	JSR \$DD8D	in buffer
DD62	A5 81	LDA \$81	sector # of the first data block
DD64	20 8D DD	JSR \$DD8D	in buffer
DD67	20 6C DE	JSR \$DE6C	write block to disk
DD6A	20 99 D5	JSR \$D599	and check
DD6D	A9 02	LDA #\$02	
DD6F	20 C8 D4	JSR \$D4C8	buffer pointer to 2
DD72	A6 82	LDX \$82	channel number
DD74	38	SEC	
DD75	A9 00	LDA #\$00	
DD77	F5 C7	SBC \$C7,X	record length
DD79	95 C1	STA \$C1,X	pointer for writing
DD7B	20 E2 E2	JSR \$E2E2	erase buffer
DD7E	20 19 DE	JSR \$DE19	write link bytes in buffer
DD81	20 5E DE	JSR \$DE5E	write block to disk
DD84	20 99 D5	JSR \$D599	and check
DD87	20 F4 FE	JSR \$EEF4	write BAM
DD8A	4C 98 DC	JMP \$DC98	and done

# Anatomy of the 1541 Disk Drive

```

***** write byte in side-sector block
DD8D  48          PHA          save byte
DD8E  A6 82       LDX $82      channel number
DD90  B5 CD       LDA $CD,X    buffer # of the side-sector
DD92  4C FD CF    JMP SCFFD    write byte in buffer

***** manipulate flags
DD95  90 06       BCC $DD9D
DD97  A6 82       LDX $82      channel number
DD99  15 EC       ORA $EC,X    set flag
DD9B  D0 06       BNE $DDA3
DD9D  A6 82       LDX $82      channel number
DD9F  49 FF       EOR #$FF
DDA1  35 EC       AND $EC,X    erase flag
DDA3  95 EC       STA $EC,X
DDA5  60          RTS
DDA6  A6 82       LDX $82      channel number
DDA8  35 EC       AND $EC,X    test flag
DDAA  60          RTS

***** check command code for writing
DDAB  20 93 DF    JSR $DF93    get buffer number
DDAE  AA          TAX
DDAF  BD 5B 02    LDA $025B,X
ddb2  29 F0       AND #$F0     isolate command code
ddb4  C9 90       CMP #$90     code for writing?
ddb6  60          RTS

*****
ddb7  A2 00       LDX #$00
ddb9  86 71       STX $71      counter for secondary address
ddbB  BD 2B 02    LDA $022B,X  get channel number from table
ddbE  C9 FF       CMP #$FF
ddc0  D0 08       BNE $DDCA    file open?
ddc2  A6 71       LDX $71
ddc4  E8          INX          increment counter
ddc5  E0 10       CPX #$10     smaller than 16?
ddc7  90 F0       BCC $DDB9
ddc9  60          RTS

DDCA  86 71       STX $71
DDCC  29 3F       AND #$3F     isolate channel number
DDCE  A8          TAY
DDCF  B9 EC 00    LDA $00EC,Y
DDD2  29 01       AND #$01     isolate drive number
DDD4  85 70       STA $70
DDD6  AE 53 02    LDX $0253
DDD9  B5 E2       LDA $E2,X
DDDB  29 01       AND #$01     isolate drive number
DDDd  C5 70       CMP $70      same drive?
DDDF  D0 E1       BNE $DDC2    no
DDE1  B9 60 02    LDA $0260,Y  sector number in directory
DDE4  D5 D8       CMP $D8,X    same as file?
DDE6  D0 DA       BNE $DDC2    no

```

# Anatomy of the 1541 Disk Drive

DDE8	B9 66 02	LDA \$0266,Y	
DDEB	D5 DD	CMP \$DD,X	pointer same?
DDED	D0 D3	BNE \$DDC2	no
DDEF	18	CLC	
DDF0	60	RTS	
***** write a block of a rel-file			
DDF1	20 9E DF	JSR \$DF9E	get buffer number
DDF4	50 06	BVC \$DDFC	no rel-file?
DDF6	20 5E DE	JSR \$DE5E	write block
DDF9	20 99 D5	JSR \$D599	and verify
DDFC	60	RTS	
***** write bytes for following track			
DDFD	20 2B DE	JSR \$DE2B	set buffer pointer
DE00	A5 80	LDA \$80	track number
DE02	91 94	STA (\$94),Y	in buffer
DE04	C8	INY	
DE05	A5 81	LDA \$81	sector number
DE07	91 94	STA (\$94),Y	in buffer
DE09	4C 05 E1	JMP \$E105	set rel-flag
***** get following track and sector #			
DE0C	20 2B DE	JSR \$DE2B	set buffer pointer
DE0F	B1 94	LDA (\$94),Y	following track number
DE11	85 80	STA \$80	
DE13	C8	INY	
DE14	B1 94	LDA (\$94),Y	and get sector number
DE16	85 81	STA \$81	
DE18	RTS		
***** following track for last block			
DE19	20 2B DE	JSR \$DE2B	set buffer pointer
DE1C	A9 00	LDA #\$00	zero
DE1E	91 94	STA (\$94),Y	as track number
DE20	C8	INY	
DE21	A6 82	LDX \$82	channel number
DE23	B5 C1	LDA \$C1,X	pointer in block
DE25	AA	TAX	
DE26	CA	DEX	minus 1
DE27	8A	TXA	
DE28	91 94	STA (\$94),Y	as pointer in block
DE2A	60	RTS	
***** buffer pointer to zero			
DE2B	20 93 DF	JSR \$DF93	get buffer number
DE2E	0A	ASL A	times 2
DE2F	AA	TAX	
DE30	B5 9A	LDA \$9A,X	buffer pointer h1
DE32	85 95	STA \$95	
DE34	A9 00	LDA #\$00	
DE36	85 94	STA \$94	buffer pointer lo
DE38	A0 00	LDY #\$00	
DE3A	60	RTS	

# Anatomy of the 1541 Disk Drive

```
*****
DE3B  20 EB D0    JSR $D0EB    get track and sector
DE3E  20 93 DF    JSR $DF93    get channel number
DE41  85 F9       STA $F9      get buffer number
DE43  0A         ASL A         save
DE44  A8         TAY          times 2
DE45  B9 06 00    LDA $0006,Y  get track
DE48  85 80       STA $80
DE4A  B9 07 00    LDA $0007,Y  and sector # from disk controller
DE4D  85 81       STA $81
DE4F  60         RTS
```

```
*****
DE50  A9 90       LDA #$90     command code for writing
DE52  8D 4D 02    STA $024D
DE55  D0 28       BNE $DE7F

DE57  A9 80       LDA #$80     command code for reading
DE59  8D 4D 02    STA $024D
DE5C  D0 21       BNE $DE7F

DE5E  A9 90       LDA #$90     command code for writing
DE60  8D 4D 02    STA $024D
DE63  D0 26       BNE $DE8B

DE65  A9 80       LDA #$80     command code for reading
DE67  8D 4D 02    STA $024D
DE6A  D0 1F       BNE $DE8B

DE6C  A9 90       LDA #$90     command code for writing
DE6E  8D 4D 02    STA $024D
DE71  D0 02       BNE $DE75

DE73  A9 80       LDA #$80     command code for reading
DE75  8D 4D 02    STA $024D
DE78  A6 82       LDX $82      channel number
DE7A  B5 CD       LDA $CD,X    side-sector buffer number
DE7C  AA         TAX
DE7D  10 13       BPL $DE92    buffer associated?
DE7F  20 D0 D6    JSR $D6D0    generate header for disk cont.
DE82  20 93 DF    JSR $DF93    get buffer number
DE85  AA         TAX
DE86  A5 7F       LDA $7F      drive number
DE88  9D 5B 02    STA $025B,X  buffer number
DE8B  20 15 E1    JSR $E115    get buffer number
DE8E  20 93 DF    JSR $DF93    get buffer number
DE91  AA         TAX
DE92  4C 06 D5    JMP $D506    write block
```

```
*****
DE95  A9 00       LDA #$00     get following track & sector from
DE97  20 C8 D4    JSR $D4C8    buffer
DE9A  20 37 D1    JSR $D137    buffer pointer to zero
DE9D  85 80       STA $80      get byte
DE9F  20 37 D1    JSR $D137    save as track
DEA2  85 81       STA $81      get byte
                                as sector
```

DEA4	60	RTS	
*****			copy buffer contents
DEA5	48	PHA	
DEA6	A9 00	LDA #\$00	
DEA8	85 6F	STA \$6F	
DEAA	85 71	STA \$71	
DEAC	B9 E0 FE	LDA \$FEE0,Y	buffer address Y, hi
DEAF	85 70	STA \$70	
DFB1	BD E0 FE	LDA \$FEE0,X	buffer address X, hi
DEB4	85 72	STA \$72	
DEB6	68	PLA	
DEB7	A8	TAY	
DEB8	88	DEY	
DEB9	B1 6F	LDA (\$6F),Y	copy contents of buffer Y
DEBB	91 71	STA (\$71),Y	to buffer X
DEBD	88	DEY	
DEBE	10 F9	BPL \$DEB9	
DEC0	60	RTS	
*****			erase buffer Y
DEC1	A8	TAY	buffer number
DEC2	B9 E0 FE	LDA \$FEE0,Y	get hi-address
DEC5	85 70	STA \$70	
DEC7	A9 00	LDA #\$00	lo-address
DEC9	85 6F	STA \$6F	
DECB	A8	TAY	
DECC	91 6F	STA (\$6F),Y	erase buffer
DECE	C8	INY	
DECF	D0 FB	BNE \$DECC	
DED1	60	RTS	
*****			get side-sector number
DED2	A9 00	LDA #\$00	
DED4	20 DC DE	JSR \$DEDC	buffer pointer to zero
DED7	A0 02	LDY #\$02	
DED9	B1 94	LDA (\$94),Y	byte 2 contains the side-sector #
DEDB	60	RTS	
*****			set buffer ptr to side-sector
DEDC	85 94	STA \$94	pointer lo
DEDE	A6 82	LDX \$82	channel number
DEE0	B5 CD	LDA \$CD,X	buffer number
DEE2	AA	TAX	
DEE3	BD E0 FE	LDA \$FEE0,X	buffer address hi
DEE6	85 95	STA \$95	set
DEE8	60	RTS	
*****			buffer pointer for side-sector
DEE9	48	PHA	pointer in side-sector
DEEA	20 DC DE	JSR \$DEDC	set buffer pointer
DEED	48	PHA	
DEEE	8A	TXA	buffer number
DEEF	0A	ASL A	times 2
DEF0	AA	TAX	

## Anatomy of the 1541 Disk Drive

DEF1	68		PLA	buffer pointer hi
DEF2	95	9A	STA \$9A,X	
DEF4	68		PLA	buffer pointer lo
DEF5	95	99	STA \$99,X	
DEF7	60		RTS	
*****				
DEF8	20	66 DF	JSR \$DF66	get side-sector and buffer ptr
DEFB	30	0E	BMI \$DF0B	is side-sector in buffer
DEFD	50	13	BVC \$DF12	no
DEFF	A6	82	LDX \$82	ok
DF01	B5	CD	LDA \$CD,X	channel number
DF03	20	1B DF	JSR \$DF1B	buffer number
DF06	20	66 DF	JSR \$DF66	read side-sector
DF09	10	07	BPL \$DF12	and check if in buffer
DF0B	20	CB E1	JSR \$E1CB	yes?
DF0E	2C	CE FE	BIT \$FECE	get last side-sector
DF11	60		RTS	set V bit
DF12	A5	D6	LDA \$D6	
DF14	20	E9 DE	JSR \$DEE9	side-sector end pointer
DF17	2C	CD DE	BIT \$FECD	set pointer in side-sector
DF1A	60		RTS	erase V bit
*****				
DF1B	85	F9	STA \$F9	read side-sector
DF1D	A9	80	LDA #\$80	buffer number
DF1F	D0	04	BNE \$DF25	command code for reading
*****				
DF21	85	F9	STA \$F9	write side-sector
DF23	A9	90	LDA #\$90	buffer number
DF25	48		PHA	command code for writing
DF26	B5	EC	LDA \$EC,X	
DF28	29	01	AND #\$01	isolate drive number
DF2A	85	7F	STA \$7F	
DF2C	68		PLA	
DF2D	05	7F	ORA \$7F	command code plus drive number
DF2F	8D	4D 02	STA \$024D	save
DF32	B1	94	LDA (\$94),Y	track number
DF34	85	80	STA \$80	
DF36	C8		INY	
DF37	B1	94	LDA (\$94),Y	sector number
DF39	85	81	STA \$81	
DF3B	A5	F9	LDA \$F9	buffer number
DF3D	20	D3 D6	JSR \$D6D3	transmit param to disk controller
DF40	A6	F9	LDX \$F9	buffer number
DF42	4C	93 D5	JMP \$D593	transmit cmd to disk controller
*****				
DF45	A6	82	LDX \$82	set buffer pointer in side-sector
DF47	B5	CD	LDA \$CD,X	channel number
DF49	4C	EB D4	JMP \$D4EB	buffer number
*****				
DF4C	A9	78	LDA #\$78	set buffer pointer
*****				
DF4C	A9	78	LDA #\$78	calculate block # of a rel-file
				120 block ptrs per side-sector

# Anatomy of the 1541 Disk Drive

DF4E	20 5C	DF	JSR \$DF5C	add to \$70/\$71
DF51	CA		DEX	side-sector number
DF52	10 F8		BPL \$DF4C	next side-sector?
DF54	A5 72		LDA \$72	pointer value in last block
DF56	4A		LSR A	divided by 2
DF57	20 5C	DF	JSR \$DF5C	add to previous sum
DF5A	A5 73		LDA \$73	number of the side-sector block
DF5C	18		CLC	
DF5D	65 70		ADC \$70	
DF5F	85 70		STA \$70	add
DF61	90 02		BCC \$DF65	
DF63	E6 71		INC \$71	
DF65	60		RTS	

\*\*\*\*\* verify side-sector in buffer

DF66	20 D2	DE	JSR \$DED2	get side-sector number
DF69	C5 D5		CMP \$D5	= number of necessary block?
DF6B	D0 0E		BNE \$DF7B	no
DF6D	A4 D6		LDY \$D6	pointer in side-sector
DF6F	B1 94		LDA (\$94),Y	track number
DF71	F0 04		BEQ \$DF77	
DF73	2C CD	FE	BIT \$FECD	erase bits
DF76	60		RTS	
DF77	2C CF	FE	BIT \$FECF	set N-bit
DF7A	60		RTS	

DF7B	A5 D5		LDA \$D5	side-sector number
DF7D	C9 06		CMP #\$06	6 or greater?
DF7F	B0 0A		BCS \$DF8B	yes
DF81	0A		ASL A	
DF82	A8		TAY	
DF83	A9 04		LDA #\$04	
DF85	85 94		STA \$94	
DF87	B1 94		LDA (\$94),Y	track number
DF89	D0 04		BNE \$DF8F	
DF8B	2C D0	FE	BIT \$FEDO	set N and V bits
DF8E	60		RTS	

DF8F	2C CE	FE	BIT \$FECE	set V bit
DF92	60		RTS	

\*\*\*\*\* get buffer number

DF93	A6 82		LDX \$82	channel number
DF95	B5 A7		LDA \$A7,X	buffer number
DF97	10 02		BPL \$DF9B	
DF99	B5 AE		LDA \$AE,X	buffer number from second table
DF9B	29 BF		AND #\$BF	erase V bit
DF9D	60		RTS	

DF9E	A6 82		LDX \$82	channel number
DFA0	8E 57	02	STX \$0257	save
DFA3	B5 A7		LDA \$A7,X	get buffer number
DFA5	10 09		BPL \$DFB0	buffer allocated
DFA7	8A		TXA	
DFA8	18		CLC	



# Anatomy of the 1541 Disk Drive

DFAB	8D 57 02	STA \$0257	increment number by 7 and save
DFAE	B5 AE	LDA \$AE,X	buffer number from table 2
DFB0	85 70	STA \$70	
DFB2	29 1F	AND #\$1F	erase the highest 3 bits
DFB4	24 70	BIT \$70	
DFB6	60	RTS	
DFB7	AD 82	LDX \$82	channel number
DFB9	B5 A7	LDA \$A7,X	buffer number
DFBB	30 02	BMI \$DFBF	buffer free?
DFBD	B5 AE	LDA \$AE,X	buffer number from table 2
DFBF	C9 FF	CMP #\$FF	free?
DFC1	60	RTS	
DFC2	A6 82	LDX \$82	
DFC4	09 80	ORA #\$80	
DFC6	B4 A7	LDY \$A7,X	
DFC8	10 03	BPL \$DFCD	
DFCA	95 A7	STA \$A7,X	
DFCC	60	RTS	
DFCD	95 AE	STA \$AE,X	
DFCF	60	RTS	
*****			
DFD0	A9 20	LDA #\$20	get next record in rel-file
DFD2	20 9D DD	JSR \$DD9D	erase bit 5
DFD5	A9 80	LDA #\$80	
DFD7	20 A6 DD	JSR \$DDA6	test bit 7
DFDA	D0 41	BNE \$E01D	set?
DFDC	A6 82	LDX \$82	channel number
DFDE	F6 B5	INC \$B5,X	increment record number
DFE0	D0 02	BNE \$DFE4	
DFE2	F6 BB	INC \$BB,X	record number h1
DFE4	A6 82	LDX \$82	channel number
DFE6	B5 C1	LDA \$C1,X	write pointer
DFE8	F0 2E	BEQ \$E018	zero?
DFEA	20 E8 D4	JSR \$D4E8	set buffer pointer
DFED	A6 82	LDX \$82	channel number
DFEF	D5 C1	CMP \$C1,X	buffer ptr smaller than write ptr
DFE1	90 03	BCC \$DFF6	yes
DFE3	20 3C E0	JSR \$E03C	write block, read next block
DFE6	A6 82	LDX \$82	channel number
DFE8	B5 C1	LDA \$C1,X	write pointer
DFEA	20 C8 D4	JSR \$D4C8	set buffer pointer = write ptr
DFED	A1 99	LDA (\$99),X	byte from buffer
DFEF	85 85	STA \$85	put in output register
E001	A9 20	LDA #\$20	
E003	20 9D DD	JSR \$DD9D	erase bit 5
E006	20 04 E3	JSR \$E304	add record length to write ptr
E009	48	PHA	and save
E00A	90 28	RCC \$E034	not yet in last block?
E00C	A9 00	LDA #\$00	
E00E	20 F6 D4	JSR \$D4F6	get track number
E011	D0 21	BNE \$E034	does block exist?

E013	68		PLA	pointer
E014	C9 02		CMP #\$02	= 2
E016	F0 12		BEQ \$E02A	yes
E018	A9 80		LDA #\$80	
E01A	20 97 DD		JSR \$DD97	set bit 7
E01D	20 2F D1		JSR \$D12F	get byte from buffer
E020	B5 99		LDA \$99,X	buffer pointer
E022	99 44 02		STA \$0244,Y	as end pointer
E025	A9 0D		LDA #\$0D	CR
E027	85 85		STA \$85	in output register
E029	60		RTS	
E02A	20 35 E0		JSR \$E035	
E02D	A6 82		LDX \$82	channel number
E02F	A9 00		LDA #\$00	
E031	95 C1		STA \$C1,X	write pointer to zero
E033	60		RTS	
E034	68		PLA	
E035	A6 82		LDX \$82	channel number
E037	95 C1		STA \$C1,X	set write pointer
E039	4C 6E E1		JMP \$E16E	
*****				write block and read next block
E03C	20 D3 D1		JSR \$D1D3	get drive number
E03F	20 95 DE		JSR \$DE95	get track and sector number
E042	20 9E DF		JSR \$DF9E	get buffer number
E045	50 16		BVC \$E05D	no rel-file?
E047	20 5E DE		JSR \$DE5E	write block
E04A	20 1E CF		JSR \$CF1E	change buffer
E04D	A9 02		LDA #\$02	
E04F	20 C8 D4		JSR \$D4C8	buffer pointer to 2
E052	20 AB DD		JSR \$DDAB	command code for writing?
E055	D0 24		BNE \$E078	no
E057	20 57 DE		JSR \$DE57	read block
E05A	4C 99 D5		JMP \$D599	and verify
E05D	20 1E CF		JSR \$CF1E	change buffer
E060	20 AB DD		JSR \$DDAB	command code for writing?
E063	D0 06		BNE \$E068	no
E065	20 57 DE		JSR \$DE57	read block
E068	20 99 D5		JSR \$D599	and verify
E06B	20 95 DE		JSR \$DE95	get track and sector number
E06E	A5 80		LDA \$80	track
E070	F0 09		BEQ \$E07B	no following track
E072	20 1E CF		JSR \$CF1E	change buffer
E075	20 57 DE		JSR \$DE57	read block
E078	20 1E CF		JSR \$CF1E	change buffer
E07B	60		RTS	
*****				write a byte in a record
E07C	20 05 E1		JSR \$E105	
E07F	20 93 DF		JSR \$DF93	get buffer number
E082	0A		ASL A	times 2
E083	AA		TAX	

# Anatomy of the 1541 Disk Drive

E084	A5 85	LDA \$85	data byte
E086	81 99	STA (\$99,X)	write in buffer
E088	B4 99	LDY \$99,X	buffer pointer
E08A	C8	INY	increment
E08B	D0 09	BNE \$E096	not equal zero?
E08D	A4 82	LDY \$82	channel number
E08F	B9 C1 00	LDA \$00C1,Y	write pointer
E092	F0 0A	BEQ \$E09E	equal zero?
E094	A0 02	LDY #\$02	buffer pointer to 2
E096	98	TYA	
E097	A5 82	LDY \$82	channel number
E099	D9 C1 00	CMP \$00C1,Y	buffer pointer = write pointer?
E09C	D0 05	BNE \$E043	no
E09E	A9 20	LDA #\$20	
E0A0	4C 97 DD	JMP \$DD97	set bit 5
E0A3	F6 99	INC \$99,X	increment buffer pointer
E0A5	D0 03	BNE \$E0AA	not zero?
E0A7	20 3C E0	JSR \$E03C	else write block, read next one
E0AA	60	RTS	
***** write byte in rel-file			
E0AB	A9 A0	LDA #\$A0	
E0AD	20 A6 DD	JSR \$DDA6	test bits 6 & 7
E0B0	D0 27	BNE \$E0D9	set?
E0B2	A5 85	LDA \$85	data byte
E0B4	20 7C E0	JSR \$E07C	write in record
E0B7	A5 F8	LDA \$F8	end?
E0B9	F0 0D	BEQ \$E0C8	yes
E0BB	60	RTS	
E0BC	A9 20	LDA #\$20	
E0BE	20 A6 DD	JSR \$DDA6	test bit 5
E0C1	F0 05	REQ \$E0C8	not set
E0C3	A9 51	LDA #\$51	51, 'overflow in record'
E0C5	8D 6C 02	STA \$026C	set error flag
E0C8	20 F3 E0	JSR \$E0F3	fill remainder with zeroes
E0CB	20 53 E1	JSR \$E153	
E0CE	AD 6C 02	LDA \$026C	error flag set?
E0D1	F0 03	BEQ \$E0D6	no
E0D3	4C C8 C1	JMP \$C1C8	set error message
E0D6	4C BC E6	JMP \$F6BC	error free execution
E0D9	29 80	AND #\$80	bit 7 set?
E0DB	D0 05	BNE \$E0E2	yes
E0DD	A5 F8	LDA \$F8	
E0DF	F0 DB	BEQ \$E0BC	end?
E0E1	60	RTS	
E0E2	A5 85	LDA \$85	data byte
E0E4	48	PHA	
E0E5	20 1C E3	JSR \$E31C	expand side-sector
E0E8	68	PLA	
E0E9	85 85	STA \$85	
E0EB	A9 80	LDA #\$80	

# Anatomy of the 1541 Disk Drive

E0ED	20 9D DD	JSR \$DD9D	erase bit 7
E0F0	4C B2 E0	JMP \$E0B2	write byte in file
*****			fill record with zeroes
E0F3	A9 20	LDA #\$20	
E0F5	20 A6 DD	JSR \$DDA6	test bit 5
E0F8	D0 0A	BNE \$E104	set?
E0FA	A9 00	LDA #\$00	
E0FC	85 85	STA \$85	zero as data byte
E0FE	20 7C E0	JSR \$E07C	write in record
E101	4C F3 E0	JMP \$E0F3	until record full
E104	60	RTS	
*****			write buffer number in table
E105	A9 40	LDA #\$40	
E107	20 97 DD	JSR \$DD97	set bit 6
E10A	20 9E DF	JSR \$DF9E	get buffer number
E10D	09 40	ORA #\$40	set bit 6
E10F	AE 57 02	LDX \$0257	channel number + 7
E112	95 A7	STA \$A7,X	write in table
E114	60	RTS	
E115	20 9E DF	JSR \$DF9E	get buffer number
E118	29 BF	AND #\$BF	erase bit 6
E11A	AE 57 02	LDX \$0257	channel number
E11D	95 A7	STA \$A7,X	write in table
E11F	60	RTS	
*****			get byte from rel-file
E120	A9 80	LDA #\$80	
E122	20 A6 DD	JSR \$DDA6	test bit 7
E125	D0 37	BNE \$E15E	set?
E127	20 2F D1	JSR \$D12F	get byte from buffer
E12A	B5 99	LDA \$99,X	buffer pointer
E12C	D9 44 02	CMP \$0244,Y	compare to end pointer
E12F	F0 22	BEQ \$E135	equal?
E131	F6 99	INC \$99,X	increment buffer pointer
E133	D0 06	BNE \$E13B	not zero?
E135	20 3C E0	JSR \$E03C	write block, read next one
E138	20 2F D1	JSR \$D12F	get byte from buffer
E13B	A1 99	LDA (\$99,X)	
E13D	99 3E 02	STA \$023E,Y	in output register
E140	A9 89	LDA #\$89	
E142	99 F2 00	STA \$00F2,Y	set READ and WRITE flag
E145	B5 99	LDA \$99,Y	buffer pointer
E147	D9 44 02	CMP \$0244,Y	compare to end pointer
E14A	F0 01	BEQ \$E14D	same?
E14C	60	RTS	
E14D	A9 81	LDA #\$81	
E14F	99 F2 00	STA \$00F2,Y	set flag for end
E152	60	RTS	
E153	20 D0 DF	JSR \$DFD0	find next record

# Anatomy of the 1541 Disk Drive

E156	20 2F D1	JSR \$D12F	get buffer and channel number
E159	A5 85	LDA \$85	data byte
E15B	4C 3D E1	JMP \$E13D	into output register
E15E	A6 82	LDX \$82	channel number
E160	A9 0D	LDA #\$0D	CR
E162	9D 3E 02	STA \$023E,X	into output register
E165	A9 81	LDA #\$81	
E167	95 F2	STA \$F2,X	set flag for end
E169	A9 50	LDA #\$50	
E16B	20 C8 C1	JSR \$C1C8	50, 'record not present'
E16E	A6 82	LDX \$82	channel number
E170	B5 C1	LDA \$C1,X	write pointer
E172	85 87	STA \$87	save
E174	C6 87	DEC \$87	
E176	C9 02	CMP #\$02	equal 2?
E178	D0 04	BNE \$E17E	no
E17A	A9 FF	LDA #\$FF	
E17C	85 87	STA \$87	
E17E	B5 C7	LDA \$C7,X	record length
E180	85 88	STA \$88	
E182	20 E8 D4	JSR \$D4E8	set buffer pointer
E185	A6 82	LDX \$82	channel number
E187	C5 87	CMP \$87	buffer pointer > write pointer?
E189	90 19	BCC \$E1A4	
E18B	F0 17	BEQ \$E1A4	no
E18D	20 1E CF	JSR \$CF1E	change buffer
E190	20 B2 E1	JSR \$E1B2	
E193	90 08	BCC \$E19D	
E195	A6 82	LDX \$82	channel number
E197	9D 44 02	STA \$0244,X	
E19A	4C 1E CF	JMP \$CF1E	change buffer
E19D	20 1E CF	JSR \$CF1E	change buffer
E1A0	A9 FF	LDA #\$FF	
E1A2	85 87	STA \$87	
E1A4	20 B2 E1	JSR \$E1B2	
E1A7	B0 03	BCS \$E1AC	
E1A9	20 E8 D4	JSR \$D4E8	set buffer pointer
E1AC	A6 82	LDX \$82	channel number
E1AE	9D 44 02	STA \$0244,X	end pointer
E1B1	60	RTS	
E1B2	20 2B DE	JSR \$DE2B	buffer pointer to zero
E1B5	A4 87	LDY \$87	
E1B7	B1 94	LDA (\$94),Y	byte from buffer
E1B9	D0 0D	BNE \$E1C8	not zero?
E1BB	88	DEY	
E1BC	C0 02	CPY #\$02	
E1BE	90 04	BCC \$E1C4	
E1C0	C6 88	DEC \$88	
E1C2	D0 F3	BNE \$E1B7	
E1C4	C6 88	DEC \$88	
E1C6	18	CLC	

# Anatomy of the 1541 Disk Drive

```
E1C7    60          RTS
E1C8    98          TYA
E1C9    38          SEC
E1CA    60          RTS
```

```
***** get last side-sector
E1CB    20 D2 DE    JSR $DED2    get number of the side-sector
E1CE    85 D5      STA $D5      save
E1D0    A9 04      LDA #$04
E1D2    85 94      STA $94      pointer to side-sectors
E1D4    A0 0A      LDY #$0A
E1D6    D0 04      BNE $E1DC
```

```
E1D8    88          DEY
E1D9    88          DEY
E1DA    30 26      BMI $E202
E1DC    B1 94      LDA ($94),Y    track # of the previous block
E1DE    F0 F8      BEQ $E1D8
E1E0    98          TYA
E1E1    4A          LSR A        divide by 2
E1E2    C5 D5      CMP $D5      = number of the actual block?
E1E4    F0 09      BEQ $E1EF    yes
E1E6    85 D5      STA $D5      else save all numbers
E1E8    A6 82      LDX $82      channel number
E1EA    B5 CD      LDA $CD,X    buffer number
E1EC    20 1B DF    JSR $DF1B    read block
E1EF    A0 00      LDY #$00
E1F1    84 94      STY $94      buffer pointer
E1F3    B1 94      LDA ($94),Y    track number
E1F5    D0 0B      BNE $E202    another block?
E1F7    C8          INY
E1F8    B1 94      LDA ($94),Y    sector number = end pointer
E1FA    A8          TAY
E1FB    88          DEY
E1FC    84 D6      STY $D6      save end pointer
E1FE    98          TYA
E1FF    4C E9 DE    JMP $DEE9    set buffer pointer
```

```
E202    A9 67      #$67
E204    20 45 E6    JSR $E645    67, 'illegal track or sector'
```

```
***** P-command, 'Record'
E207    20 B3 C2    JSR $C2B3    verify lines
E20A    AD 01 02    LDA $0201    secondary address
E20D    85 83      STA $83
E20F    20 EB D0    JSR $D0EB    find channel number
E212    90 05      BCC $E219    found?
E214    A9 70      LDA #$70
E216    20 C8 C1    JSR $C1C8    70, 'no block'
```

```
E219    A9 A0      LDA #$A0
E21B    20 9D DD    JSR $DD9D    erase bits 6 & 7
E21E    20 25 D1    JSR $D125    verify if 'REL'-file
E221    F0 05      BEQ $E228    yes
```

# Anatomy of the 1541 Disk Drive

E223	A9	64		LDA #\$64	
E225	20	C8	C1	JSR \$C1C8	64, 'file type mismatch'
E228	B5	EC		LDA \$EC,X	
E22A	29	01		AND #\$01	
E22C	85	7F		STA \$7F	drive number
E22E	AD	02	02	LDA \$0202	record number lo
E231	95	B5		STA \$B5,X	
E233	AD	03	02	LDA \$0203	record number hi
E236	95	BB		STA \$BB,X	
E238	A6	B2		LDA \$82	channel number
E23A	A9	89		LDA #\$89	
E23C	95	F2		STA \$F2,X	READ and WRITE flag
E23E	AD	04	02	LDA \$0204	byte-pointer
E241	F0	10		BEQ \$E253	zero?
E243	38			SEC	
E244	E9	01		SBC #\$01	
E246	F0	0B		BEQ \$E253	
E248	D5	C7		CMP \$C7,X	compare with record length
E24A	90	07		BCC \$E253	
E24C	A9	51		LDA #\$51	
E24E	8D	6C	02	STA \$026C	51, 'overflow in record'
E251	A9	00		LDA #\$00	
E253	85	D4		STA \$D4	
E255	20	0E	CE	JSR \$CE0E	calculate pointer in rel-file
E258	20	F8	DE	JSR \$DEF8	and read appropriate side-sector
E25B	50	08		BVC \$E265	does block exist?
E25D	A9	80		LDA #\$80	
E25F	20	97	DD	JSR \$DD97	set bit 7
E262	4C	5E	E1	JMP \$E15E	and 50, 'record not present'
E265	20	75	E2	JSR \$E275	
E268	A9	80		LDA #\$80	
E26A	20	A6	DD	JSR \$DDA6	test bit 7
E26D	F0	03		BEQ \$E272	not set
E26F	4C	5E	E1	JMP \$E15E	50, 'record not present'
E272	4C	94	C1	JMP \$C194	done
E275	20	9C	E2	JSR \$E29C	
E278	A5	D7		LDA \$D7	pointer in rel-file
E27A	20	C8	D4	JSR \$D4C8	set buffer pointer
E27D	A6	82		LDX \$82	channel number
E27F	B5	C7		LDA \$C7,X	record length
E281	38			SEC	
E282	E5	D4		SBC \$D4	minus position
E284	B0	03		BCS \$E289	positive?
E286	4C	02	E2	JMP \$E202	67, 'illegal track or sector'
E289	18			CLC	
E28A	65	D7		ADC \$D7	add pointer in data block
E28C	90	03		BCC \$E291	no overflow
E28E	69	01		ADC #\$01	plus 2
E290	38			SEC	
E291	20	09	E0	JSR \$E009	set pointer
E294	4C	38	E1	JMP \$E138	get byte from buffer

# Anatomy of the 1541 Disk Drive

E297	A9	51		LDA #\$51	
E299	20	C8	C1	JSR \$C1C8	51, 'overflow in record'
E29C	A5	94		LDA \$94	buffer pointer lo
E29E	85	89		STA \$89	
E2A0	A5	95		LDA \$95	buffer pointer hi
E2A2	85	8A		STA \$8A	
E2A4	20	D0	E2	JSR \$E2D0	compare track and sector
E2A7	D0	01		BNE \$E2AA	not equal?
E2A9	60			RTS	
E2AA	20	F1	DD	JSR \$DDF1	
E2AD	20	0C	DE	JSR \$DE0C	
E2B0	A5	80		LDA \$80	track
E2B2	F0	0E		BEQ \$E2C2	no block following?
E2B4	20	D3	E2	JSR \$E2D3	compare track and sector number
E2B7	D0	06		BNE \$E2BF	not equal?
E2B9	20	1E	CF	JSR \$CF1E	change buffer
E2BC	4C	DA	D2	JMP \$D2DA	
E2BF	20	DA	D2	JSR \$D2DA	
E2C2	A0	00		LDY #\$00	
E2C4	B1	89		LDA (\$89),Y	track
E2C6	85	80		STA \$80	
E2C8	C8			INY	
E2C9	B1	89		LDA (\$89),Y	and sector of the next block
E2CB	85	81		STA \$81	
E2CD	4C	AF	D0	JMP \$D0AF	read block
E2D0	20	3E	DE	JSR \$DE3E	
E2D3	A0	00		LDY #\$00	
E2D5	B1	89		LDA (\$89),Y	track number
E2D7	C5	80		CMP \$80	compare
E2D9	F0	01		BEQ \$E2DC	
E2DB	60			RTS	
E2DC	C8			INY	
E2DD	B1	89		LDA (\$89),Y	sector number
E2DF	C5	81		CMP \$81	compare
E2E1	60			RTS	
*****					subdivide records in data block
E2E2	20	2B	DE	JSR \$DE2B	set buffer pointer
E2E5	A0	02		LDY #\$02	
E2E7	A9	00		LDA #\$00	
E2E9	91	94		STA (\$94),Y	erase buffer
E2EB	C8			INY	
E2EC	D0	FB		BNE \$E2E9	
E2EE	20	04	E3	JSR \$E304	set pointer to next record
E2F1	95	C1		STA \$C1,X	
E2F3	A8			TAY	
E2F4	A9	FF		LDA #\$FF	
E2F6	91	94		STA (\$94),Y	\$FF as 1st character in record
E2F8	20	04	E3	JSR \$E304	set pointer to next record
E2FB	90	F4		BCC \$E2F1	done in this block?
E2FD	D0	04		BNE \$E303	block full?



## Anatomy of the 1541 Disk Drive

E2FF	A9 00	LDA #\$00	
E301	95 C1	STA \$C1,X	write pointer to zero
E303	60	RTS	
***** set pointer to next record			
E304	A6 82	LDX \$82	channel number
E306	B5 C1	LDA \$C1,X	write pointer
E308	38	SEC	
E309	F0 0D	BEQ \$E318	equal zero?
E30B	18	CLC	
E30C	75 C7	ADC \$C7,X	add record length
E30E	90 0B	BCC \$E31B	smaller than 256?
E310	D0 06	BNE \$E318	equal 256?
E312	A9 02	LDA #\$02	
E314	2C CC FE	BIT \$FECC	
E317	60	RTS	
E318	69 01	ADC #\$01	add two
E31A	38	SEC	
E31B	60	RTS	
***** expand side-sector			
E31C	20 D3 D1	JSR \$D1D3	get drive number
E31F	20 CB E1	JSR \$E1CB	get last side-sector
E322	20 9C E2	JSR \$E29C	
E325	20 7B CF	JSR \$CF7B	
E328	A5 D6	LDA \$D6	
E32A	85 87	STA \$87	
E32C	A5 D5	LDA \$D5	side-sector number
E32E	85 86	STA \$86	
E330	A9 00	LDA #\$00	
E332	85 88	STA \$88	
E334	A9 00	LDA #\$00	
E336	85 D4	STA \$D4	
E338	20 0E CE	JSR \$CE0E	calculate side-sector no. and ptr
E33B	20 4D EF	JSR \$EF4D	number of free blocks
E33E	A4 82	LDY \$82	channel number
E340	B6 C7	LDX \$C7,Y	record length
E342	CA	DEX	
E343	8A	TXA	
E344	18	CLC	
E345	65 D7	ADC \$D7	plus pointer in data block
E347	90 0C	BCC \$E355	
E349	E6 D6	INC \$D6	
E34B	E6 D6	INC \$D6	increment ptr to end by 2
E34D	D0 06	BNE \$E355	
E34F	E6 D5	INC \$D5	increment side-sector number
E351	A9 10	LDA #\$10	
E353	85 D6	STA \$D6	set pointer to 16
E355	A5 87	LDA \$87	
E357	18	CLC	
E358	69 02	ADC #\$02	
E35A	20 E9 DE	JSR \$DEE9	set buffer ptr for side-sector
E35D	A5 D5	LDA \$D5	side-sector number
E35F	C9 06	CMP #\$06	

# Anatomy of the 1541 Disk Drive

E361	90 05	BCC \$E368	smaller than 6?
E363	A9 52	LDA #\$52	
E365	20 C8 C1	JSR \$C1C8	52, 'file too large'
E368	A5 D6	LDA \$D6	end pointer
E36A	38	SEC	
E36B	E5 87	SBC \$87	minus last end pointer
E36D	B0 03	BCS \$E372	
E36F	E9 0F	SBC #\$0F	minus 16
E371	18	CLC	
E372	85 72	STA \$72	
E374	A5 D5	LDA \$D5	side-sector number
E376	E5 86	SBC \$86	minus last side-sector number
E378	85 73	STA \$73	save
E37A	A2 00	LDX #\$00	
E37C	86 70	STX \$70	erase sum for calculation
E37E	86 71	STX \$71	
E380	AA	TAX	
E381	20 51 DF	JSR \$DF51	calculate block # of rel-file
E384	A5 71	LDA \$71	
E386	D0 07	BNE \$E38F	
E388	A6 70	LDX \$70	
E38A	CA	DEX	
E38B	D0 02	BNE \$E38F	
E38D	E6 88	INC \$88	
E38F	CD 73 02	CMP \$0273	block number of rel-file
E392	90 09	BCC \$E39D	greater than free blocks on disk?
E394	D0 CD	BNE \$E363	52, 'file too large'
E396	AD 72 02	LDA \$0272	
E399	C5 70	CMP \$70	
E39B	90 C6	BCC \$E363	52, 'file too large'
E39D	A9 01	LDA #\$01	
E39F	20 F6 D4	JSR \$D4F6	get byte from buffer
E3A2	18	CLC	
E3A3	69 01	ADC #\$01	plus 1
E3A5	A6 82	LDX \$82	
E3A7	95 C1	STA \$C1,X	as write pointer
E3A9	20 1E F1	JSR \$F11E	find free block in RAM
E3AC	20 FD DD	JSR \$DDFD	track and sector in buffer
E3AF	A5 88	LDA \$88	
E3B1	D0 15	BNE \$E3C8	only one block needed?
E3B3	20 5E DE	JSR \$DE5E	write block
E3B6	20 1E CF	JSR \$CF1E	change buffer
E3B9	20 D0 D6	JSR \$D6D0	transmit param to disk controller
E3BC	20 1E F1	JSR \$F11E	find free block in RAM
E3BF	20 FD DD	JSR \$DDFD	track and sector in buffer
E3C2	20 E2 E2	JSR \$E2E2	erase buffer
E3C5	4C D4 E3	JMP \$E3D4	
E3C8	20 1E CF	JSR \$CF1E	change buffer
E3CB	20 D0 D6	JSR \$D6D0	transmit param to disk controller
E3CE	20 E2 E2	JSR \$E2E2	erase buffer
E3D1	20 19 DE	JSR \$DE19	zero byte and end ptr in buffer
E3D4	20 5E DE	JSR \$DE5E	write block
E3D7	20 0C DE	JSR \$DE0C	get track and sector
E3DA	A5 80	LDA \$80	track

# Anatomy of the 1541 Disk Drive

E3DC	48		PHA	
E3DD	A4 81		LDA \$81	and sector
E3DF	48		PHA	save
E3E0	20 3E DE		JSR \$DE3E	get track and sector from disk
E3E3	A5 81		LDA \$81	controller
E3E5	48		PHA	
E3E6	A5 80		LDA \$80	save track and sector
E3E8	48		PHA	
E3E9	20 45 DF		JSR \$DF45	set buffer ptr for side-sector
E3EC	AA		TAX	
E3ED	D0 0A		BNE \$E3F9	pointer not zero?
E3EF	20 4E E4		JSR \$E44E	write side-sector
E3F2	A9 10		LDA #\$10	
E3F4	20 E9 DE		JSR \$DEE9	buffer pointer to 16
E3F7	E6 86		INC \$86	increment side-sector number
E3F9	68		PLA	
E3FA	20 8D DD		JSR \$DD8D	track in side sector
E3FD	68		PLA	
E3FE	20 8D DD		JSR \$DD8D	sector in side-sector
E401	68		PLA	
E402	85 81		STA \$81	sector
E404	68		PLA	
E405	85 80		STA \$80	and get track back
E407	F0 0F		BEQ \$E418	no more blocks?
E409	A5 86		LDA \$86	side-sector number
E40B	C5 D5		CMP \$D5	changed?
E40D	D0 A7		BNE \$E3B6	yes
E40F	20 45 DF		JSR \$DF45	set buffer ptr in side-sector
E412	C5 D6		CMP \$D6	end pointer
E414	90 A0		BCC \$E3B6	smaller?
E416	F0 R0		BEQ \$E3C8	same
E418	20 45 DF		JSR \$DF45	set buffer ptr in side-sector
E41B	48		PHA	
E41C	A9 00		LDA #\$00	
E41E	20 DC DE		JSR \$DEDC	buffer pointer to zero
E421	A9 00		LDA #\$00	
E423	A8		TAY	
E424	91 94		STA (\$94),Y	zero as track number
E426	C8		INY	
E427	68		PLA	end pointer
E428	38		SEC	
E429	E9 01		SBC #\$01	minus one
E42B	91 94		STA (\$94),Y	as sector
E42D	20 6C DE		JSR \$DE6C	write block
E430	20 99 D5		JSR \$D599	and verify
E433	20 F4 EE		JSR \$EEF4	update BAM
E436	20 0E CE		JSR \$CE0E	update pointer for rel-file
E439	20 1E CF		JSR \$CF1E	change buffer
E43C	20 F8 DE		JSR \$DEF8	right side-sector?
E43F	70 03		BVS \$E444	no
E441	4C 75 E2		JMP \$E275	
E444	A9 80		LDA #\$80	
E446	20 97 DD		JSR \$DD97	set bit 7
E449	A9 50		LDA #\$50	

# Anatomy of the 1541 Disk Drive

E44B	20 C8 C1	JSR \$C1C8	50, 'record not present'
*****			write side-sector and allocate new one
E44E	20 1E F1	JSR \$F11E	find free block in BAM
E451	20 1E CF	JSR \$CF1E	change buffer
E454	20 F1 DD	JSR \$DDF1	write block
E457	20 93 DF	JSR \$DF93	get buffer number
E45A	48	PHA	
E45B	20 C1 DE	JSR \$DEC1	erase buffer
E45E	A6 82	LDX \$82	channel number
E460	B5 CD	LDA \$CD,X	buffer number
E462	A8	TAY	
E463	68	PLA	
E464	AA	TAX	
E465	A9 10	LDA #\$10	16 bytes of the side-sector
E467	20 A5 DE	JSR \$DEA5	copy in buffer
E46A	A9 00	LDA #\$00	
E46C	20 DC DE	JSR \$DEDC	buffer ptr to 0, old side-sector
E46F	A0 02	LDY #\$02	
E471	B1 94	LDA (\$94),Y	side-sector number
E473	48	PHA	
E474	A9 00	LDA #\$00	
E476	20 C8 D4	JSR \$D4C8	buffer ptr to 0, new side-sector
E479	68	PLA	
E47A	18	CLC	
E47B	69 01	ADC #\$01	increment side-sector number
E47D	91 94	STA (\$94),Y	and in buffer
E47F	0A	ASL A	times 2
E480	69 04	ADC #\$04	plus 4
E482	85 89	STA \$89	
E484	A8	TAY	
E485	38	SEC	
E486	E9 02	SBC #\$02	minus 2
E488	85 8A	STA \$8A	same pointer to old side-sector
E48A	A5 80	LDA \$80	track
E48C	85 87	STA \$87	
E48E	91 94	STA (\$94),Y	in buffer
E490	C8	INY	
E491	A5 81	LDA \$81	sector
E493	85 88	STA \$88	
E495	91 94	STA (\$94),Y	in buffer
E497	A0 00	LDY #\$00	
E499	98	TYA	
E49A	91 94	STA (\$94),Y	zero in buffer
E49C	C8	INY	
E49D	A9 11	LDA #\$11	17
E49F	91 94	STA (\$94),Y	number of bytes in block
E4A1	A9 10	LDA #\$10	16
E4A3	20 C8 D4	JSR \$D4C8	buffer pointer to 16
E4A6	20 50 DE	JSR \$DE50	write block
E4A9	20 99 D5	JSR \$D599	and verify
E4AC	A6 82	LDX \$82	channel number
E4AE	B5 CD	LDA \$CD,X	buffer number of the side-sector
E4B0	48	PHA	

# Anatomy of the 1541 Disk Drive

E4B1	20 9E DF	JSR \$DF9E	get buffer number
E4B4	A6 82	LDX \$82	channel number
E4B6	95 CD	STA \$CD,X	write in table
E4B8	68	PLA	
E4B9	AE 57 02	LDX \$0257	channel number + 7
E4BC	95 A7	STA \$A7,X	in table
E4BE	A9 00	LDA #\$00	
E4C0	20 C8 D4	JSR \$D4C8	buffer pointer to zero
E4C3	A0 00	LDY #\$00	
E4C5	A5 80	LDA \$80	track
E4C7	91 94	STA (\$94),Y	in buffer
E4C9	C8	INY	
E4CA	A5 81	LDA \$81	sector
E4CC	91 94	STA (\$94),Y	in buffer
E4CE	4C DE E4	JMP \$E4DE	
E4D1	20 93 DF	JSR \$DF93	get buffer number
E4D4	A6 82	LDX \$82	channel number
E4D6	20 1B DF	JSR \$DF1B	read block
E4D9	A9 00	LDA #\$00	
E4DB	20 C8 D4	JSR \$D4C8	buffer pointer to zero
EFDE	C6 8A	DEC \$8A	
E4E0	C6 8A	DEC \$8A	counter for side-sector blocks
E4E2	A4 89	LDY \$89	
E4E4	A5 87	LDA \$87	track number
E4E6	91 94	STA (\$94),Y	in buffer
E4E8	C8	INY	
E4E9	A5 88	LDA \$88	sector number
E4EB	91 94	STA (\$94),Y	in buffer
E4ED	20 5E DE	JSR \$DE5E	write block
E4F0	20 99 D5	JSR \$D599	and verify
E4F3	A4 8A	LDY \$8A	counter for side-sector blocks
E4F5	C0 03	CPY #\$03	
E4F7	B0 D8	BCS \$E4D1	greater than or equal to 3?
E4F9	4C 1E CF	JMP \$CF1E	change buffer

*****	table of error messages
E4FC 00	00
E4FD A0 4F CB	' OK'
E500 20 21 22 23 24 27	error numbers of 'read error'
E506 D2 45 41 44	'Read'
E50A 89	pointer to 'error'
E50B 52	52
E50C 83	pointer to 'file'
E50D 20 54 4F 4F 20 AC 4A 52 47	C5 ' too large'
E517 50	50
E518 8B 06	pointer to 'record ' and 'not '
E51A 20 50 52 45 53 45 4E D4	' present'
E522 51	51
E523 CF 56 45 52 46 4C 4F 57 20	'Overflow in'
E52E 8B	pointer to 'record'
E52F 25 28	error numbers of 'write error'
E531 8A 89	pointer to 'write' and 'error '
E533 26	26
E534 8A	pointer to 'write'

# Anatomy of the 1541 Disk Drive

E535	20	50	52	4F	54	45	43	54	20	4F CE	'protect oN'
E540	29									29	
E541	88									pointer to 'disk'	
E542	20	49	85							'id'	
E545	85									pointer to 'mismatch'	
E546	30	31	32	33	34					error numbers for 'syntax error'	
E54B	D3	59	4E	54	41	58				'Syntax'	
E551	89									pointer to 'error'	
E552	60									60	
E553	8A	03	84							ptrs to 'write', 'file' & 'open'	
E556	63									63	
E557	83									pointer to 'file'	
E558	20	45	58	49	53	54	D3			'exists'	
E55F	64									64	
E560	83									pointer to 'file'	
E561	20	54	59	50	45					'type'	
E566	85									pointer to 'mismatch'	
E567	65									65	
E568	CE	4F	20	42	4C	4F	43	CB		'No block'	
E570	66	67								'illegal track or sector'	
E572	C9	4C	4C	45	47	41	4C	20		'Illegal'	
E57A	54	52	41	43	4B	20	4F	52		'track or'	
E582	20	53	45	43	54	4F	D2			'sector'	
E589	61									61	
E58A	83	06	84							pointer to 'file', 'not' & 'open'	
E58D	39	62								error nos. for 'file not found'	
E590	83	06	87							ptrs to 'file', 'not' & 'found'	
E593	01									01	
E594	83									pointer to 'file'	
E594	53	20	53	43	52	41	54	43	48	45 C4 's scratched'	
E59F	70									70	
ESA0	CE	4F	20	43	48	41	4E	4E	45	CC 'No channel'	
ESAA	71									71	
ESAB	C4	49	52							'Dir'	
ESAE	89									pointer to 'error'	
ESAF	72									72	
ESB0	88									pointer to 'disk'	
ESB1	20	46	55	4C	CC					'full'	
ESB6	73									73	
ESB7	C3	42	4D	20	44	4F	53	20		'Cbm dos'	
ESBF	56	32	2E	36	20	31	35	34	B1	'v2.6 1541'	
ESC4	74									74	
ESC5	C4	42	49	56	45					'Drive'	
ESCA	06									pointer to 'not'	
ESCB	20	52	45	41	44	D9				'ready'	
ESD5	09										
ESD6	C5	52	52	4F	D2					'Error'	
ESDB	0A										
ESDC	D7	52	49	54	C5					'Write'	
ESE1	03										
ESE2	C6	49	4C	C5						'File'	
ESE6	04										
E6E7	CF	50	45	CE						'Open'	
E5EB	05										
E5EC	CD	49	53	4D	41	54	43	C8		'Mismatch'	

# Anatomy of the 1541 Disk Drive

```

E5F4 06
E5F5 CE 4F D4      'NoT'
E5F8 07
E5F9 C6 4F 55 4E C4 'Found'
E5FE 08
E5FF C4 49 53 CB   'DisK'
E603 0B
E604 D2 45 43 4F 52 C4 'Record'
```

```

***** prepare error number and message
E60A 48 PHA save error code
E60B 86 F9 STX $F9 drive number
E60D 8A TXA
E60E 0A ASL A times 2
E60F AA TAX as pointer
E610 B5 06 LDA $06,X
E612 85 80 STA $80 get track
E614 B5 07 LDA $07,X
E616 85 81 STA $81 and sector number
E618 68 PLA get error code back
E619 29 0F AND #$0F isolate bits 0-3
E61B F0 08 BEQ $E625 zero, then 24, 'read error'
E61D C9 0F CMP #$0F 15?
E61F D0 06 BNE $E627
E621 A9 74 LDA #$74
E623 D0 08 BNE $E62D 74, 'drive not ready'
E625 A9 06 LDA #$06 6
E627 09 20 ORA #$20 add $20
E629 AA TAX
E62A CA DEX
E62B CA DEX subtract two
E62C 8A TXA
E62D 48 PHA save error number
E62E AD 2A 02 LDA $022A number of the disk command
E631 C9 00 CMP #$00 OPEN or VALIDATE?
E633 D0 0F BNE $E644 no
E635 A9 FF LDA #$FF
E637 8D 2A 02 STA $022A
E63A 68 PLA get error number back
E63B 20 C7 E6 JSR $E6C7 generate error message
E63E 20 42 D0 JSR $D042 load BAM
E641 4C 48 E6 JMP $E648 set error message

E644 68 PLA
E645 20 C7 E6 JSR $E6C7 set error message
E648 20 BD C1 JSR $C1BD erase input buffer
E64B A9 00 LDA #$00
E64D 8D F9 02 STA $02F9 erase error flag
E650 20 2C C1 JSR $C12C turn LED off
E653 20 DA D4 JSR $D4DA close channels 17 and 18
E656 A9 00 LDA #$00
E658 85 A3 STA $A3 input buffer pointer to zero
E65A A2 45 LDX #$45
E65C 9A TXS initialize stack pointer
E65D A5 84 LDA $84 secondary address
```

# Anatomy of the 1541 Disk Drive

```

E65F 29 0F      AND #$0F
E661 85 83      STA $83
E663 C9 0F      CMP #$0F      15?
E665 F0 31      BEQ $E698      yes, command channel
E667 78         SEI
E668 A5 79      LDA $79        LISTEN active?
E66A D0 1C      BNE $E688      yes
E66C A5 7A      LDA $7A        TALK active?
E66E D0 10      BNE $E680      yes
E670 A6 83      LDX $83        channel number
E672 BD 2B 02   LDA $022B,X    open channel to this second. addr
E675 C9 FF      CMP #$FF
E677 F0 1F      BEQ $E698      no
E679 29 0F      AND #$0F
E67B 85 82      STA $82        channel number
E67D 4C 8E E6   JMP $E68E

***** TALK
E680 20 EB D0   JSR $D0EB      open channel for reading
E683 20 4E EA   JSR $EA4E      accept byte
E686 D0 06      BNE $E68E

***** LISTEN
E688 20 07 D1   JSR $D107      open channel for writing
E68B 20 4E EA   JSR $EA4E      accept byte
E68E 20 25 D1   JSR $D125      verify file type
E691 C9 04      CMP #$04       file type REL?
E693 B0 03      BCS $E698      yes
E695 20 27 D2   JSR $D227      close channel
E698 4C E7 EB   JMP $EBE7

***** convert hex to decimal (2 bytes)
E69B AA         TAX
E69C A9 00      LDA #$00
E69E F8         SED
E69F E0 00      CPX #$00
F6A1 F0 07      BEQ $E6AA      convert hex to BCD
E6A3 18         CLC
E6A4 69 01      ADC #$01
A6A6 CA         DEX
E6A7 4C 9F E6   JMP $E69F
E6AA D8         CLD

***** divide BCD number into two bytes
E6AB AA         TAX
E6AC 4A         LSR A
E6AD 4A         LSR A      shift hi-nibble down
E6AE 4A         LSR A
E6AF 4A         LSR A
E6B0 20 B4 E6   JSR $E6B4      convert to ASCII
E6B3 8A         TXA
E6B4 29 0F      AND #$0F      erase top 4 bits
E6B6 09 30      ORA #$30      add '0'
E6B8 91 A5      STA ($A5),Y    write in buffer
E6BA C8         INY            increment buffer pointer

```



# Anatomy of the 1541 Disk Drive

E6BB 60 RTS

```
***** write 'ok' in buffer
E6BC 20 23 C1 JSR $C123 erase error flag
E6BF A9 00 LDA #$00 error number 0
E6C1 A0 00 LDY #$00
E6C3 84 80 STY $80 track 0
E6C5 84 81 STY $81 sector 0
```

```
***** error message in buffer
E6C7 A0 00 LDY #$00 buffer pointer
E6C9 A2 D5 LDX #$D5
E6C8 86 A5 STX $A5 pointer $A5/$A6 TO $2D5
E6CD A2 02 LDX #$02
E6CF 86 A6 STX $A6
E6D1 20 AB E6 JSR $E6AB error # to ASCII and in buffer
E6D4 A9 2C LDA #$2C ',' comma
E6D6 9A A5 STA ($A5),Y write in buffer
E6D8 C8 INY increment buffer pointer
E6D9 AD D5 02 LDA $02D5 first digit of the disk status
E6DC 8D 43 02 STA $0243 in output register
E6DF 8A TXA error number in accumulator
E6E0 20 06 E7 JSR $E706 error message in buffer
E6E3 A9 2C LDA #$2C ',' comma
E6E5 91 A5 STA ($A5),Y write in buffer
E6E7 C8 INY and increment buffer pointer
E6F8 A5 80 LDA $80 track number
E6EA 20 9B E6 JSR $E69B to ASCII and in buffer
E6ED A9 2C LDA #$2C ',' comma
E6EF 91 A5 STA ($A5),Y write in buffer
E6F1 C8 INY increment buffer pointer
E6F2 A5 81 LDA $81 sector
E6F4 20 9B E6 JSR $E69B convert to ASCII and in buffer
E6F7 88 DEY
E6F8 98 TYA
E6F9 18 CLC
E6FA 69 D5 ADC #$D5
E6FC 8D 49 02 STA $0249 end pointer
E6FF E6 A5 INC $A5
E701 A9 88 LDA #$88 set READ flag
E703 85 F7 STA $F7
E705 60 RTS
```

```
***** write error message to buffer
E706 AA TAX error code to X
E707 A5 86 LDA $86
E709 48 PHA preserve pointer $86/$87
E70A A5 87 LDA $87
E70C 48 PHA
E70D A9 FC LDA #$FC
F70F 85 86 STA #$E4 start of the error messages
E713 85 87 STA $87
E715 8A TXA error number in accumulator
E716 A2 00 LDX #$00
E718 C1 86 CMP ($86,X) compare with error no in table
```

## Anatomy of the 1541 Disk Drive

E71A	F0 21	BEQ \$E73D	
E71C	48	PHA	
E71D	20 75 E7	JSR \$E775	bit 7 into carry and erase
E720	90 05	BCC \$E727	not set?
E722	20 75 E7	JSR \$E775	bit 7 into carry
E725	90 FB	BCC \$E722	wait for character with bit 7 set
E727	A5 87	LDA \$87	
E729	C9 E6	CMP #\$E6	
E72B	90 08	BCC \$E735	\$E60A, check to end of table
E72D	D0 0A	BNE \$E739	
E72F	A0 0A	LDA #\$0A	
E731	C5 86	CMP \$86	
E733	90 04	BCC \$E739	
E735	68	PLA	
E736	4C 18 E7	JMP \$E718	no, continue
E739	68	PLA	
E73A	4C 4D E7	JMP \$E74D	done
E73D	20 67 E7	JSR \$E767	get a character, bit 7 in carry
E740	90 FB	BCC \$E73D	wait for character with bit 7 set
E742	20 54 E7	JSR \$E754	and write in buffer
E745	20 67 E7	JST \$E767	get next character
E748	90 F8	BCC \$E742	wait for character with bit 7 set
E74A	20 54 E7	JSR \$E754	put character in buffer
E74D	68	PLA	
E74E	85 87	STA \$87	
E750	68	PLA	get pointer \$86/\$87 back
E751	85 86	STA \$86	
E753	60	RTS	
*****		get character and in buffer	
E754	C9 20	CMP #\$20	' ' blank
E756	B0 0B	BCS \$E763	greater, then write in buffer
E758	AA	TAX	save code
E759	A9 20	LDA #\$20	blank
E75B	91 A5	STA (\$A5),Y	write in buffer
E75D	C8	INY	increment buffer pointer
E75E	8A	TXA	code in accumulator
E75F	20 06 E7	JSR \$E706	output previous text
E762	60	RTS	
E763	91 A5	STA (\$A5),Y	write character in buffer
E765	C8	INY	and increment pointer
E766	60	RTS	
*****		get a char of the error message	
E767	E6 86	INC \$86	
E769	D0 02	BNE \$E76D	increment pointer
E76B	E6 87	INC \$87	
E76D	A1 86	LDA (\$86,X)	get character
E76F	0A	ASL A	bit 7 into carry
E770	A1 86	LDA (\$86,X)	get character
E772	29 7F	AND #\$7F	erase bit 7
E774	60	PTS	
*****		increment pointer	

# Anatomy of the 1541 Disk Drive

E775	20 6D E7	JSR \$E76D	bit 7 into carry
E778	E6 86	INC \$86	
E77A	D0 02	BNE \$E77E	increment pointer
E77C	E6 87	INC \$87	
E77E	60	RTS	

\*\*\*\*\*

E77F	60	RTS
------	----	-----

*****			check for AUTO-start
E780	AD 00 18	LDA \$1800	read IEEE port
E783	AA	TAX	
E784	29 04	AND #\$04	isolate 'CLOCK IN' bit
E786	F0 F7	BEQ \$E77F	not set, then done
E788	8A	TXA	
E789	29 01	AND #\$01	isolate 'DATA IN' bit
E78B	F0 F2	BEQ \$E77F	not set, then done
E78D	58	CLI	
E78E	AD 00 18	LDA \$1800	load IEEE port
E791	29 05	AND #\$05	test 'DATA IN' and 'CLOCK IN'
E793	F0 F9	BNE \$E78E	wait until both set
E795	EE 78 02	INC \$0278	file name
E798	EE 74 02	INC \$0274	character in the input line
E79B	A9 2A	LDA #\$2A	'*' as filename
E79D	8D 00 02	STA \$0200	write in buffer
E7A0	4C A8 E7	JMP \$E7A8	

\*\*\*\*\* 'g' - command

E7A3	A9 8D	LDA #\$8D	
E7A5	20 68 C2	JSR \$C268	check command line to end
E7A8	20 58 F2	JSR \$F258	(RTS)
E7AB	AD 78 02	LDA \$0278	number of file names
E7AE	48	PHA	save
E7AF	A9 01	LDA #\$01	
E7B1	8D 78 02	STA \$0278	file name
E7B4	A9 FF	LDA #\$FF	
E7B6	85 86	STA \$86	
E7B8	20 4F C4	JSR \$C44F	find file
E7BB	AD 80 02	LDA \$0280	
E7BE	D0 05	BNE \$E7C5	found?
E7C0	A9 39	LDA #\$39	
E7C2	20 C8 C1	JSR \$C1C8	39, 'file not found'
E7C5	68	PLA	
E7C6	8D 78 02	STA \$0278	get number of file names back
E7C9	AD 80 02	LDA \$0280	
E7CC	85 80	STA \$80	track
E7CE	AD 85 02	LDA \$0285	
E7D1	85 81	STA \$81	and sector
E7D3	A9 03	LDA #\$03	file type 'USR'
E7D5	20 77 D4	JSR \$D477	buffer allocated, read 1st block
E7D8	A9 00	LDA #\$00	
E7DA	85 87	STA \$87	erase checksum
E7DC	20 39 E8	JSR \$E839	get byte from file
E7DF	85 88	STA \$88	save as start address lo
E7E1	20 4B E8	JSR \$E84B	form checksum

# Anatomy of the 1541 Disk Drive

E7E4	20 39 E8	JSR \$E839	get byte from file
E7E7	85 89	STA \$89	as start address hi
E7E9	20 4B E8	JSR \$E84B	form checksum
E7EC	A5 86	LDA \$86	
E7EE	F0 0A	BEQ \$E7FA	
E7F0	A5 88	LDA \$88	
E7F2	48	PHA	save program start address
E7F3	A5 89	LDA \$89	
E7F5	48	PHA	
E7F6	A9 00	LDA #\$00	
E7F8	85 86	STA \$86	
E7FA	20 39 E8	JSR \$E839	get byte from file
E7FD	85 8A	STA \$8A	save as counter
E7FF	20 4B E8	JSR \$E84B	form checksum
E802	20 39 E8	JSR \$E839	get byte from file
E805	A0 00	LDY #\$00	
E807	91 88	STA (\$88),Y	save as program bytes
E809	20 4B E8	JSR \$E84B	form checksum
E80C	A5 88	LDA \$88	
E80E	18	CLC	
E80F	69 01	ADC #\$01	
E811	85 88	STA \$88	increment \$88/\$89
E813	90 02	BCC \$E817	
E815	E6 89	INC \$89	
E817	C6 8A	DEC \$8A	decrement pointer
E819	D0 E7	BNE \$E802	
E81B	20 35 CA	JSR \$CA35	get next byte
E81E	A5 85	LDA \$85	data byte
E820	C5 87	CMP \$87	equal to checksum?
E822	F0 08	BEQ \$E82C	yes
E824	20 3E DE	JSR \$DE3E	transmit param to disk controller
E827	A9 50	LDA #\$50	
E829	20 45 E6	JSR \$E645	50, 'record not present'
E82C	A5 F8	LDA \$F8	end?
E82E	D0 A8	BNE \$E7D8	no, next data block
E830	68	PLA	
E831	85 89	STA \$89	
E833	68	PLA	get program start address back
E834	85 88	STA \$88	
E836	6C 88 00	JMP (\$0088)	and execute program
E839	20 35 CA	JSR \$CA35	get byte from file
E83C	A5 F8	LDA \$F8	end?
E83E	D0 08	BNE \$E848	no
E840	20 3E DE	JSR \$DE3E	transmit param to disk controller
E843	A9 51	LDA #\$51	
E845	20 45 E6	JSR \$E645	51, 'overflow in record'
E848	A5 85	LDA \$85	data byte
E84A	60	RTS	
*****			generate checksum
E84B	A8	CLC	
E84C	65 87	ADC \$87	
E84E	69 00	ADC #\$00	
E850	85 87	STA \$87	
E852	60	RTS	

# Anatomy of the 1541 Disk Drive

```

*****
E853  AD 01 18  LDA $1801  IRQ routine for serial bus
E856  A9 01      LDA #$01  read port A, erase IRQ flag
E858  85 7C      STA $7C   set flag for 'ATN received'
E85A  60          RTS

*****
E85B  78          SEI      servicing the serial bus
E85C  A9 00      LDA #$00
E85E  85 7C      STA $7C   erase flag for 'ATN received'
E860  85 79      STA $79   erase flag for LISTEN
E862  85 7A      STA $7A   erase flag for TALK
E864  A2 45      LDX #$45
E866  9A          TXS      initialize stack pointer
E867  A9 80      LDA #$80
E869  85 F8      STA $F8   erase end flag
E86B  85 7D      STA $7D   erase EOF flag
E86D  20 B7 E9   JSR $E9B7  CLOCK OUT lo
E870  20 A5 E9   JSR $E9A5  DATA OUT, bit '0', hi
E873  AD 00 18  LDA $1800
E876  09 10      ORA #$10  switch data lines to input
E878  8D 00 18  STA $1800
E87B  AD 00 18  LDA $1800  read IEEE port
E87E  10 57      BPL $E8D7  EOF?
E880  29 04      AND #$04  CLOCK IN?
E882  D0 F7      BNE $E87B  no
E884  20 C9 E9   JSR $E9C9  get byte from bus
E887  C9 3F      CMP #$3F  unlisten?
E889  D0 06      BNE $E891  no
E88B  A9 00      LDA #$00
E88D  85 79      STA $79   reset flag for LISTEN
E88F  F0 71      BEQ $E902
E891  C9 5F      CMP #$5F  untalk?
E893  D0 06      BNE $E89B  no
E895  A9 00      LDA #$00
E897  85 7A      STA $7A   reset flag for TALK
E899  F0 67      BEQ $E902
E89B  C5 78      CMP $78   TALK address?
E89D  D0 0A      BNE $E8A9  no
E89F  A9 01      LDA #$01
E8A1  85 7A      STA $7A   set flag for TALK
E8A3  A9 00      LDA #$00
E8A5  85 79      STA $79   reset flag for LISTEN
E8A7  F0 29      BEQ $E8D2
E8A9  C5 77      CMP $77   LISTEN address?
E8AB  D0 0A      BNE $E8B7  no
E8AD  A9 01      LDA #$01
E8AF  85 79      STA $79   set flag for LISTEN
E8B1  A9 00      LDA #$00
E8B3  85 7A      STA $7A   reset flag for TALK
E8B5  F0 1B      BEQ $E8D2
E8B7  AA          TAX
E8B8  29 60      AND #$60
E8BA  C9 60      CMP #$60  set bit 5 and 6

```

# Anatomy of the 1541 Disk Drive

E8BC	D0 3F	BNE \$E8FD	no
E8BE	8A	TXA	
E8BF	85 84	STA \$84	byte is secondary address
E8C1	29 0F	AND #\$0F	
E8C3	85 83	STA \$83	channel number
E8C5	A5 84	LDA \$84	
E8C7	29 F0	AND #\$F0	
E8C9	C9 E0	CMP #\$E0	CLOSE?
E8CB	D0 35	BNE \$E902	
E8CD	58	CLI	
E8CE	20 C0 DA	JSR \$DAC0	CLOSE routine
E8D1	78	SEI	
E8D2	2C 00 18	BIT \$1800	
E8D5	30 AD	BMI \$E884	
E8D7	A9 00	LDA #\$00	
E8D9	85 7D	STA \$7D	set EOI
E8DB	AD 00 18	LDA \$1800	IEEE port
E8DE	29 EF	AND #\$EF	switch data lines to output
E8E0	8D 00 18	STA \$1800	
E8E3	A5 79	LDA \$79	LISTEN active?
E8E5	F0 06	BEQ \$E8ED	no
E8E7	20 2E EA	JSR \$EA2E	receive data
E8EA	4C E7 EB	JMP \$EBE7	to delay loop
E8ED	A5 7A	LDA \$7A	TALK active?
E8EF	F0 09	BEQ \$E8FA	no
E8F1	20 9C E9	JSR \$E99C	DATA OUT, bit '1', 10
E8F4	20 AE E9	JSR \$E9AE	CLOCK OUT hi
E8F7	20 09 E9	JSR \$E909	send data
E8FA	4C 4E EA	JMP \$EA4E	to delay loop
E8FD	A9 10	LDA #\$10	either TALK or LISTEN, ignore byte
E8FF	8D 00 18	STA \$1800	switch data lines to input
E902	2C 00 18	BIT \$1800	
E905	10 D0	BPL \$E8D7	
E907	30 F9	BMI \$E902	wait for handshake
*****			send data
E909	78	SEI	
E90A	20 EB D0	JSR \$D0EB	open channel for read
E90D	B0 06	BCS \$E915	channel active
E90F	A6 82	LDX \$82	channel number
E911	B5 F2	LDA \$F2,X	set READ flag?
E913	30 01	BMI \$E916	yes
E915	60	RTS	
E916	20 59 FA	JSR \$EA59	check EOI
E919	20 C0 E9	JSR \$E9C0	read IEEE port
E91C	29 01	AND #\$01	isolate data bit
E91E	08	PHP	and save
E91F	20 B7 E9	JSR \$E9B7	CLOCK OUT lo
E922	28	PLP	
E923	F0 12	BEQ \$E937	
E925	20 59 EA	JSR \$EA59	check EOI
E928	20 C0 E9	JSR \$E9C0	read IEEE port
E92B	29 01	AND #\$01	isolate data bit
E92D	D0 F6	BNE \$E925	

# Anatomy of the 1541 Disk Drive

E92F	A6 82	LDX \$82	channel number
E931	B5 F2	LDA \$F2,X	
E933	29 08	AND #\$08	
E935	D0 14	BNE \$E94B	
E937	20 59 EA	JSR \$EA59	check EOI
E93A	20 C0 E9	JSR \$E9C0	read IEEE port
E93D	29 01	AND #\$01	isolate data bit
E93F	D0 F6	BNE \$E937	
E941	20 59 EA	JSR \$EA59	check EOI
E944	20 C0 E9	JSR \$E9C0	read IEEE port
E947	29 01	AND #\$01	isolate data bit
E949	F0 F6	BEQ \$E941	
E84B	20 AE E9	JSR \$E9AE	CLOCK OUT hi
E94E	20 59 EA	JSR \$EA59	check EOI
E951	20 C0 E9	JSR \$E9C0	read IEEE port
E954	29 01	AND #\$01	isolate data bit
E956	D0 F3	BNE \$E94B	
E958	A9 08	LDA #\$08	counter to 8 bits for serial
E95A	85 98	STA \$98	transmission
E95C	20 C0 E9	JSR \$E9C0	read IEEE port
E95F	29 01	AND #\$01	isolate data bit
E961	D0 36	BNE \$E999	
E963	A6 82	LDX \$82	
E965	BD 3E 02	LDA \$023E,X	
E968	6A	ROR A	lowest bit in carry
E969	9D 3E 02	STA \$023E,X	
E96C	B0 05	BCS \$E973	set bit
E96E	20 A5 E9	JSR \$E9A5	DATA OUT, output bit '0'
E971	D0 03	BNE \$E976	absolute jump
E973	20 9C E9	JSR \$E99C	DATA OUT, output bit '1'
E976	20 B7 E9	JSR \$E9B7	set CLOCK OUT
E979	A5 23	LDA \$23	
E97B	D0 03	BNE \$E980	
E97D	20 F3 FE	JSR \$FEF3	delay for serial bus
E980	20 FB FE	JSR \$FEFB	set DATA OUT and CLOCK OUT
E983	C6 98	DEC \$98	all bits output?
E985	D0 D5	BNE \$E95C	no
E987	20 59 EA	JSR \$EA59	check EOI
E98A	20 C0 E9	JSR \$E9C0	read IEEE port
E98D	29 01	AND #\$01	isolate data bit
E98F	F0 F6	BEQ \$E987	
E991	58	CLI	
E992	20 AA D3	JSR \$D3AA	get next data byte
E995	78	SEI	
E996	4C 0F E9	JMP \$E90F	and output
E999	4C 4E EA	JMP \$EA4E	to delay loop
*****			DATA OUT lo
E99C	AD 00 18	LDA \$1800	
E99F	29 FD	AND #\$FD	output bit '1'
E9A1	8D 00 18	STA \$1800	
E9A4	60	RTS	
*****			DATA OUT hi

# Anatomy of the 1541 Disk Drive

E9A5	AD 00 18	LDA \$1800	
E9A8	09 02	ORA #\$02	output bit '0'
E9AA	8D 00 18	STA \$1800	
E9AD	60	RTS	
*****			
E9AE	AD 00 18	LDA \$1800	CLOCK OUT hi
E9B1	09 08	ORA #\$08	set bit 3
E9B3	8D 00 18	STA \$1800	
E9B6	60	RTS	
*****			
E9B7	AD 00 18	LDA \$1800	CLOCK OUT lo
E9BA	29 F7	AND #\$F7	erase bit 3
E9BC	8D 00 18	STA \$1800	
E9BF	60	RTS	
*****			
E9C0	AD 00 18	LDA \$1800	read IEEE port
E9C3	CD 00 18	CMP \$1800	read port
E9C6	D0 F8	BNE \$E9C0	wait for constants
E9C8	60	RTS	
*****			
E9C9	A9 08	LDA #\$08	
E9CB	85 98	STA \$98	bit counter for serial output
E9CD	20 59 EA	JSR \$EA59	check EOI
E9D0	20 C0 E9	JSR \$E9C0	read IEEE port
E9D3	29 04	AND #\$04	CLOCK IN?
E9D5	D0 F6	BNE \$E9CD	no, wait
E9D7	20 9C E9	JSR \$E99C	DATA OUT, bit '1'
E9DA	A9 01	LDA #\$01	
E9DC	8D 05 18	STA \$1805	set timer
E9DF	20 59 EA	JSR \$EA59	check EOI
E9E2	AD 0D 18	LDA \$180D	
E9E5	29 40	AND #\$40	timer run down?
E9E7	D0 09	BNE \$E9F2	yes, EOI
E9E9	20 C0 E9	JSR \$E9C0	read IEEE port
E9EC	29 04	AND #\$04	CLOCK IN?
E9EE	F0 EF	BEQ \$E9DF	no, wait
E9F0	D0 19	BNE \$EA0B	
E9F2	20 A5 E9	JSR \$E9A5	DATA OUT bit '0' hi
E9F5	A2 0A	LDY #\$0A	lo
E9F7	CA	DEX	delay loop, approx 50 micro sec.
E9F8	D0 FD	BNE \$E9F7	
E9FA	20 9C E9	JSR \$E99C	DATA OUT, bit '1', lo
E9FD	20 59 EA	JSR \$EA59	check EOI
EA00	20 C0 E9	JSR \$E9C0	read IEEE
EA03	29 04	AND #\$04	CLOCK IN?
EA05	F0 F6	BEQ \$E9FD	no, wait
EA07	A9 00	LDA #\$00	
EA09	85 F8	STA \$F8	set EOI flag
EA0B	AD 00 18	LDA \$1800	IEEE port
EA0E	49 01	EOR #\$01	invert data byte
EA10	4A	LSR A	



# Anatomy of the 1541 Disk Drive

EA11	29 02	AND #\$02	
EA13	D0 F6	BNE \$EA0B	CLOCK IN?
EA15	EA	NOP	
EA16	EA	NOP	
EA17	EA	NOP	
EA18	66 85	ROR \$85	prepare next bit
EA1A	20 59 EA	JSR \$EA59	check EOI
EA1D	20 C0 E9	JSR \$E9C0	read IEEE port
EA20	29 04	AND #\$04	CLOCK IN?
EA22	F0 F6	BEQ \$EA1A	no
EA24	C6 98	DEC \$98	decrement bit counter
EA26	D0 E3	BNE \$EA0B	all bits output?
EA28	20 A5 E9	JSR \$E9A5	DATA OUT, bit '0', hi
EA2B	A5 85	LDA \$85	load data byte again
EA2D	60	RTS	
***** accept data from serial bus			
EA2E	78	SEI	
EA2F	20 07 D1	JSR \$D107	open channel for writing
EA32	B0 05	RCS \$EA39	channel not active?
EA34	B5 F2	LDA \$F2,X	WRITE flag
EA36	6A	ROR A	
EA37	B0 0B	BCS \$EA44	not set?
EA39	A5 84	LDA \$84	secondary address
EA3B	29 F0	AND #\$F0	
EA3D	C9 F0	CMP #\$F0	OPEN command?
EA3F	F0 03	BEQ \$EA44	yes
EA41	4C 4E EA	JMP \$EA4E	to wait loop
EA44	20 C9 E9	JSR \$E9C9	get data byte from bus
EA47	58	CLI	
EA48	20 B7 CF	JSR \$CFB7	and write in buffer
EA4B	4C 2E EA	JMP \$EA2E	to loop beginning
EA4E	A9 00	LDA #\$00	
EA50	8D 00 18	STA \$1800	reset IEEE port
EA53	4C E7 EB	JMP \$EBE7	to wait loop
EA56	4C 5B E8	JMP \$EB58	to serial bus main loop
*****			
EA59	A5 7D	LDA \$7D	EOI received?
EA5B	F0 06	BEO \$EA63	yes
EA5D	AD 00 18	LDA \$1800	IEEE port
EA60	10 09	BPL \$EA6B	
EA62	60	RTS	
EA63	AD 00 18	LDA \$1800	IEEE port
EA66	10 FA	BPL \$EA62	
EA68	4C D7 E8	JMP \$E8D7	set EOI, serve serial bus
***** blink LED for hardware defects			
EA6E	A2 00	LDX #\$00	blink once, zero page
EA70	2C	.BYTE \$2C	

## Anatomy of the 1541 Disk Drive

EA71	A5 6F	LDX \$6F	blink X+1 times for RAM/ROM err
EA73	9A	TXS	
EA74	BA	TSX	
EA75	A9 08	LDA #\$08	select LED bit in the port
EA77	0D 00 1C	ORA \$1C00	
EA7A	4C EA FE	JMP \$FEEA	turn LED on, back to \$EA7D
EA7D	98	TYA	
EA7E	18	CLC	
EA7F	69 01	ADC #\$01	
EA81	D0 FC	BNE \$EA7F	
EA83	88	DEY	
EA84	D0 F8	BNE \$EA7E	
EA86	AD 00 1C	LDA \$1C00	
EA89	29 F7	AND #\$F7	turn LED off
EA8B	8D 00 1C	STA \$1C00	
EA8E	98	TYA	
EA8F	18	CLC	
EA90	69 01	ADC #\$01	
EA92	D0 FC	BNE \$EA90	delay loop
EA94	88	DEY	
EA95	D0 F8	BNE \$EA8F	
EA97	CA	DEX	
EA98	10 DB	BPL \$EA75	
EA9A	E0 FC	CPX #\$FC	
EA9C	D0 F0	BNE \$EA8E	wait for delay
EA9E	F0 D4	BEQ \$EA74	turn LED on again

*****			RESET routine
EAA0	78	SEI	
EAA1	D8	CLD	
FAA2	A2 FF	LDX #\$FF	
EAA4	8E 03 18	STX \$1803	port A to output
EAA7	E8	INX	
EAA8	A0 00	LDY #\$00	
EAAA	A2 00	LDX #\$00	
EAAC	8A	TXA	
EAAD	95 00	STA \$00,X	erase zero page
EAAF	E8	INX	
EAB0	D0 FA	BNE \$EAAC	
EAB2	8A	TXA	
EAB3	D5 00	CMP \$00,X	is byte erased?
EAB5	D0 B7	BNE \$EA6E	no, then to error display (blink)
EAB7	F6 00	INC \$00,X	
EAB9	C8	INY	
EABA	D0 FB	BNE \$EAB7	
EABC	D5 00	CMP \$00,X	
EABE	D0 AE	BNE \$EA6E	error
EAC0	94 00	STY \$00,X	
EAC2	B5 00	LDA \$00,X	
EAC4	D0 A8	BNE \$EA6E	error
EAC6	E8	INX	
EAC7	D0 E9	BNE \$EAB2	
EAC9	E6 6F	INC \$6F	
EACB	86 76	STX \$76	
EACD	A9 00	LDA #\$00	

# Anatomy of the 1541 Disk Drive

EACF	85 75	STA \$75	
EAD1	A8	TAY	
EAD2	A2 20	LDX #\$20	test 32 pages
EAD4	18	CLC	
EAD5	C6 76	DEC \$76	
EAD7	71 75	ADC (\$75),Y	
EAD9	C8	INY	
EADA	D0 FB	BNE \$EAD7	
EADC	CA	DEX	
EADD	D0 F6	BNE \$EAD5	test ROM
EADF	69 00	ADC #\$00	
EAE1	AA	TAX	
EAE2	C5 76	CMP \$76	
EAE4	D0 39	BNE \$EB1F	ROM error
EAE6	E0 C0	CPX #\$C0	
EAE8	D0 DF	BNE \$EAC9	
EAEA	A9 01	LDA #\$01	
EAEC	85 76	STA \$76	
EAEE	E6 6F	INC \$6F	
EAFO	A2 07	LDX #\$07	test RAM, beginning at page 7
FAF2	98	TYA	
FAF3	18	CLC	
FAF4	65 76	ADC \$76	
FAF6	91 75	STA (\$75),Y	
FAF8	C8	INY	
FAF9	D0 F7	BNE \$EAF2	
FAFB	E6 76	INC \$76	
FAFD	CA	DEX	
FAFE	D0 F2	BNE \$EAF2	
EB00	A2 07	LDX #\$07	
EB02	C6 76	DEC \$76	
EB04	88	DEY	
EB05	98	TYA	
EB06	18	CLC	
EB07	65 76	ADC \$76	
EB09	D1 75	CMP (\$75),Y	
EB0B	D0 12	BNE \$EB1F	RAM error
EB0D	49 FF	EOR #\$FF	
EB0F	91 75	STA (\$75),Y	
EB11	51 75	EOR (\$75),Y	
EB13	91 75	STA (\$75),Y	
EB15	D0 08	BNE \$EB1F	RAM error
EB17	98	TYA	
EB18	D0 EA	BNE \$EB04	
EB1A	CA	DEX	
EB1B	D0 E5	BNE \$EB02	continue test
EB1D	F0 03	BEQ \$EB22	ok
EB1F	4C 71 EA	JMP \$EA71	to error display
EB22	A2 45	LDX #\$45	
EB24	9A	TXS	initialize stack pointer
EB25	AD 00 1C	LDA \$1C00	
EB28	29 F7	AND #\$F7	turn LED off
EB2A	8D 00 1C	STA \$1C00	
EB2D	A9 01	LDA #\$01	

## Anatomy of the 1541 Disk Drive

EB2F	8D 0C 18	STA \$180C	CA1 (ATN IN) trigger on pos edge
EB32	A9 82	LDA #\$82	
EB34	8D 0D 18	STA \$180D	interrupt possible through ATN IN
EB37	8D 0E 18	STA \$180E	
EB3A	AD 00 18	LDA \$1800	read port B
EB3D	29 60	AND #\$60	isolate bits 5 & 6 (device #)
EB3F	0A	ASL A	
EB40	2A	ROL A	
EB41	2A	ROL A	rotate to bit positions 0 & 1
EB42	2A	ROL A	
EB43	09 48	ORA #\$48	add offset from 8 + \$40 for TALK
EB45	85 78	STA \$78	device number for TALK (send)
EB47	49 60	EOR #\$60	erase bit 6, set bit 5
EB49	85 77	STA \$77	device number + \$20 for LISTEN
EB4B	A2 00	LDX #\$00	
EB4D	A0 00	LDY #\$00	
EB4F	A9 00	LDA #\$00	
EB51	95 99	STA \$99,X	low-byte of buffer address
EB53	E8	INX	
EB54	B9 E0 FE	LDA \$FEE0,Y	high byte of address from table
EB57	95 99	STA \$99,X	save
EB59	E8	INX	
EB5A	C8	INY	
EB5B	C0 05	CPY #\$05	
EB5D	D0 F0	BNE \$EB4F	
EB5F	A9 00	LDA #\$00	
EB61	95 99	STA \$99,X	
EB63	E8	INX	ptr \$A3/\$A4 to \$200, input buffer
EB64	A9 02	LDA #\$02	
EB66	95 99	STA \$99,X	
EB68	E8	INX	
EB69	A9 D5	LDA #\$D5	
EB6B	95 99	STA \$99,X	
EB6D	E8	INX	pointer \$A5/\$A6 to \$2D5, error
EB6E	A9 02	LDA #\$02	message pointer
EB70	95 99	STA \$99,X	
EB72	A9 FF	LDA #\$FF	
EB74	A2 12	LDX #\$12	
EB76	9D 2B 02	STA \$022B,X	fill channel table with \$FF
EB79	CA	DEX	
EB7A	10 FA	BPL \$EB76	
EB7C	A2 05	LDX #\$05	
EB7E	95 A7	STA \$A7,X	erase buffer table
EB80	95 AE	STA \$AE,X	
EB82	95 CD	STA \$CD,X	erase side-sector table
EB84	CA	DEX	
EB85	10 F7	BPL \$EB7E	
EB87	A9 05	LDA #\$05	buffer 5
EB89	85 AB	STA \$AB	associate with channel 4
EB8B	A9 06	LDA #\$06	buffer 6
EB8D	85 AC	STA \$AC	associate with channel 5
EB8F	A9 FF	LDA #\$FF	
EB91	85 AD	STA \$AD	
EB93	85 B4	STA \$B4	
EB95	A9 05	LDA #\$05	

# Anatomy of the 1541 Disk Drive

EB97	8D 3B 02	STA \$023B	channel 5 WRITE flag erased
EB9A	A9 84	LDA #\$84	
EB9C	8D 3A 02	STA \$023A	channel 4 WRITE flag set
EB9F	A9 0F	LDA #\$0F	initialize channel allocation reg
EBA1	8D 56 02	STA \$0256	bit '1' equals channel free
EBA4	A9 01	LDA #\$01	
EBA6	85 F6	STA \$F6	WRITE flag
EBA8	A9 88	LDA #\$88	
EBAA	85 F7	STA \$F7	READ flag
EBAC	A9 F0	LDA #\$E0	5 buffers free
EBAE	8D 4F 02	STA \$024F	initialize buffer allocation reg
EBB1	A9 FF	LDA \$FFF	\$24F/\$250, 16 bit
EBB3	8D 50 02	STA \$0250	
EBB6	A9 01	LDA #\$01	
EBB8	85 1C	STA \$1C	flags for WRITE protect
EBBA	85 1D	STA \$1D	
EBBC	20 63 CB	JSR \$CB63	set vector for U0
EBBF	20 FA CE	JSR \$CEFA	initialize channel table
EBC2	20 59 F2	JSR \$F259	intialization for disk controller
EBC5	A9 22	LDA #\$22	
EBC7	85 65	STA \$65	
EBC9	A9 EB	LDA \$EB	pointer \$65/\$66 to \$EB22
EBCB	85 66	STA \$66	
EBCD	A9 0A	LDA \$0A	
EBCF	85 69	STA \$69	step width 10
EBD1	A9 05	LDA #\$05	for sector assignment
EBD3	85 6A	STA \$6A	5 read attempts
EBD5	A9 73	LDA #\$73	prepare power-up message
EBD7	20 C1 E6	JSR \$E6C1	73, 'cbm dos v2.6 1541'
EBDA	A9 1A	LDA \$1A	bit 1, 3 & 4 to exit
EBDC	8D 02 18	STA \$1802	data direction of port B
EBDF	A9 00	LDA \$00	
EBE1	8D 00 18	STA \$1800	erase data register
EBE4	20 80 E7	JSR \$E780	check for auto-start
EBE7	58	CLI	
EBE8	AD 00 18	LDA \$1800	
EBEB	29 E5	AND \$E5	reset serial port
EBED	8D 00 18	STA \$1800	
EBF0	AD 55 02	LDA \$0255	command flag set?
EBF3	F0 0A	BEQ \$EBFF	no
EBF5	A9 00	LDA \$00	
EBF7	8D 55 02	STA \$0255	reset command flag
EBFA	85 67	STA \$67	
EBFC	20 46 C1	JSR \$C146	analyze and execute command
*****			wait loop
EBFF	58	CLI	
EC00	A5 7C	LDA \$7C	ATN signal discovered?
EC02	F0 03	BEQ \$EC07	no
EC04	4C 5B E8	JMP \$E85B	to IEEE routine
EC07	58	CLI	
EC08	A9 0E	LDA \$0E	14
EC0A	85 72	STA \$72	as secondary address
EC0C	A9 00	LDA \$00	
EC0E	85 6F	STA \$6F	job counter

# Anatomy of the 1541 Disk Drive

EC10	85 70	STA \$70	
EC12	A6 72	LDX \$72	
EC14	BD 2B 02	LDA \$022B,X	secondary address
EC17	C9 FF	CMP #\$FF	channel associated?
EC19	F0 10	BEQ \$EC2B	no
EC1B	26 3F	AND #\$3F	
EC1D	85 82	STA \$82	channel number
EC1F	20 93 DF	JSR \$DF93	get buffer number
EC22	AA	TAX	
EC23	BD 5B 02	LDA \$025B,X	drive number
EC26	29 01	AND #\$01	
EC28	AA	TAX	
EC29	F6 6F	INC \$6F,X	increment job counter
EC2B	C6 72	DEC \$72	lo address
EC2D	10 E3	BPL \$EC12	continue search
EC2F	A0 04	LDY #\$04	buffer counter
EC31	B9 00 00	LDA \$0000,Y	disk controller in action?
EC34	10 05	BPL \$EC3B	no
EC36	29 01	AND #\$01	isolate drive number
EC38	AA	TAX	
EC39	F6 6F	INC \$6F,X	increment job counter
EC3B	88	DEY	
EC3C	10 F3	BPL \$EC31	next buffer
EC3E	78	SEI	
EC3F	AD 00 1C	LDA \$1C00	
EC42	29 F7	AND #\$F7	erase LED bit
EC44	48	PHA	
EC45	A5 7F	LDA \$7F	drive number
EC47	85 86	STA \$86	
EC49	A9 00	LDA #\$00	
EC4B	85 7F	STA \$7F	drive 0
EC4D	A5 6F	LDA \$6F	job for drive 0?
EC4F	F0 0B	BEQ \$EC5C	no
EC51	A5 1C	LDA \$1C	write protect for drive 0?
EC53	F0 03	BEQ \$EC58	no
EC55	20 13 D3	JSR \$D313	close all channels to drive 0
EC58	68	PLA	
EC59	09 08	ORA #\$08	set LED bit
EC5B	48	PHA	
EC5C	E6 7F	INC \$7F	increment drive number
EC5E	A5 70	LDA \$70	job for drive 1?
EC60	F0 0B	BEQ \$EC6D	no
EC62	A5 1D	LDA \$1D	write protect for drive 1?
EC64	F0 03	BEQ \$EC69	no
EC66	20 13 D3	JSR \$D313	close all channels to drive 1
EC69	68	PLA	
EC6A	09 00	ORA #\$00	
EC6C	48	PHA	
EC6D	A5 86	LDA \$86	
EC6F	85 7F	STA \$7F	get drive number back
EC71	68	PLA	bit for LED
EC72	AE 6C 02	LDX \$026C	interrupt counter
EC75	F0 21	BEQ \$EC98	to zero?
EC77	AD 00 1C	LDA \$1C00	
EC7A	E0 80	CPX #\$80	

# Anatomy of the 1541 Disk Drive

EC7C	D0 03	BNE \$EC81	
EC7E	4C 8B EC	JMP \$EC8B	
EC81	AE 05 18	LDX \$1805	erase timer interrupt
EC84	30 12	BMI \$EC98	
EC86	A2 A0	LDX #SA0	
EC88	8E 05 18	STX \$1805	set timer
EC8B	CE 6C 02	DEC \$026C	decrement counter
EC8E	D0 08	BNE \$EC98	not yet zero?
EC90	4D 6D 02	EOR \$026D	
EC93	A2 10	LDX #\$10	
EC95	8E 6C 02	STX \$026C	reset counter
EC98	8D 00 1C	STA \$1C00	turn LED on/off
EC9B	4C FF EB	JMP \$EBFF	back to wait loop

*****			LOAD "\$"
EC9E	A9 00	LDA #\$00	
ECA0	85 83	STA \$83	secondary address 0
ECA2	A9 01	LDA #\$01	
ECA4	20 E2 D1	JSR \$D1E2	find channel and buffer
ECA7	A9 00	LDA #\$00	
ECA9	20 C8 D4	JSR \$D4C8	initialize buffer pointer
ECAC	A6 82	LDX \$82	channel number
ECAE	A9 00	LDA #\$00	
ECB0	9D 44 02	STA \$0244,X	pointer to end = zero
ECB3	20 93 DF	JSR \$DF93	get buffer number
ECB6	AA	TAX	
ECB7	A5 7F	LDA \$7F	drive number
ECB9	9D 5B 02	STA \$025B,X	bring in table
ECBC	A9 01	LDA #\$01	1
ECBE	20 F1 CF	JSR \$CFF1	write in buffer
ECB1	A9 04	LDA #\$04	4, start address \$0401
ECC3	20 F1 CF	JSR \$CFF1	write in buffer
ECC6	A9 01	LDA #\$01	2 times 1
ECC8	20 F1 CF	JSR \$CFF1	
ECCR	20 F1 CF	JSR \$CFF1	write in buffer as link address
ECCE	AD 72 02	LDA \$0272	drive number
ECD1	20 F1 CF	JSR \$CFF1	write in buffer as line number
ECD4	A9 00	LDA #\$00	line number hi
ECD6	20 F1 CF	JSR \$CFF1	in buffer
ECD9	20 59 ED	JSR \$ED59	directory entry in buffer
ECDC	20 93 DF	JSR \$DF93	get buffer number
ECDF	0A	ASL A	
ECE0	AA	TAX	
ECE1	D6 99	DEC \$99,X	decrement buffer pointer
ECE3	D6 99	DEC \$99,X	
ECE5	A9 00	LDA #\$00	
ECE7	20 F1 CF	JSR \$CFF1	0 as line end in buffer
ECEA	A9 01	LDA #\$01	
ECEC	20 F1 CF	JSR \$CFF1	2 times 1 as link address
ECEF	20 F1 CF	JSR \$CFF1	
ECF2	20 CE C6	JSR \$C6CE	directory entry in buffer
ECF5	90 2C	BCC \$ED23	another entry?
ECF7	AD 72 02	LDA \$0272	block number lo
ECFA	20 F1 CF	JSR \$CFF1	in buffer
ECFD	AD 73 02	LDA \$0273	block number hi

# Anatomy of the 1541 Disk Drive

ED00	20 F1 CF	JSR SCFF1	in buffer
ED03	20 59 ED	JSR \$ED59	directory entry in buffer
ED06	A9 00	LDA #\$00	
ED08	20 F1 CF	JSR SCFF1	zero as end marker in buffer
ED0B	D0 DD	BNE \$CECA	buffer full? no
ED0D	20 93 DF	JSR \$DF93	get buffer number
ED10	0A	ASL A	
ED11	AA	TAX	
ED12	A9 00	LDA #\$00	
ED14	95 99	STA \$99,X	buffer pointer to zero
ED16	A9 88	LDA #\$88	set READ flag
ED18	A4 82	LDY \$82	channel number
ED1A	8D 54 02	STA \$0254	
ED1D	99 F2 00	STA \$00F2,Y	flag for channel
ED20	A5 85	LDA \$85	data byte
ED22	60	RTS	

\*\*\*\*\*

ED23	AD 72 02	LDA \$0272	block number lo
ED26	20 F1 CF	JSR SCFF1	write in buffer
ED29	AD 73 02	LDA \$0273	block number hi
ED2C	20 F1 CF	JSR SCFF1	in buffer
ED2F	20 59 ED	JSR \$ED59	'Blocks free.' in buffer
ED32	20 93 DF	JSR \$DF93	get buffer number
ED35	0A	ASL A	
ED36	AA	TAX	
ED37	D6 99	DEC \$99,X	
ED39	D6 99	DEC \$99,X	buffer pointer minus 2
ED3B	A9 00	LDA #\$00	
ED3D	20 F1 CF	JSR SCFF1	
ED40	20 F1 CF	JSR SCFF1	three zeroes as program end
ED43	20 F1 CF	JSR SCFF1	
ED46	20 93 DF	JSR \$DF93	get buffer number
ED49	0A	ASL A	times 2
ED4A	A8	TAY	
ED4B	B9 99 02	LDA \$0099,Y	buffer pointer
ED4E	A6 82	LDX \$82	
ED50	9D 44 02	STA \$0244,X	as end marker
ED53	DE 44 02	DEC \$0244,X	
ED56	4C 0D ED	JMP \$ED0D	

\*\*\*\*\*

ED59	A0 00	LDY #\$00	transmit directory line
ED5B	B9 B1 02	LDA \$02B1,Y	character from buffer
ED5E	20 F1 CF	JSR SCFF1	write in output buffer
ED61	C8	INY	
ED62	C0 1B	CPY #\$1B	27 characters?
ED64	D0 F5	BNE \$ED5B	
ED66	60	RTS	

\*\*\*\*\*

ED67	20 37 D1	JSR \$D137	get byte from buffer
ED6A	F0 01	BEQ \$ED6D	get byte
ED6C	60	RTS	buffer pointer zero?



# Anatomy of the 1541 Disk Drive

ED6D	85 85	STA \$85	save data byte
ED6F	A4 82	LDY \$82	channel number
ED71	B9 44 02	LDA \$0244,Y	set end marker
ED74	F0 08	BEQ \$ED7E	zero (LOAD \$)?
ED76	A9 80	LDA #\$80	
ED78	99 F2 00	STA \$00F2,Y	set READ flag
ED7B	A5 85	LDA \$85	data byte
ED7D	60	RTS	
ED7E	48	PHA	
ED7F	20 EA EC	JSR \$ECEA	create directory line in buffer
ED82	68	PLA	
ED83	60	RTS	
*****			
			V command, 'collect'
ED84	20 D1 C1	JSR \$C1D1	find drive number in input line
ED87	20 42 D0	JSR \$D042	load BAM
ED8A	A9 40	LDA #\$40	
ED8C	8D F9 02	STA \$02F9	
ED8F	20 B7 EE	JSR \$EEB7	create new BAM in buffer
ED92	A9 00	LDA #\$00	
ED94	8D 92 02	STA \$0292	
ED97	20 AC C5	JSR \$C5AC	load directory, find 1st flag
ED9A	D0 3D	BNE \$EDD9	found?
ED9C	A9 00	LDA #\$00	
ED9E	85 81	STA \$81	sector 0
EDA0	AD 8E FE	LDA \$FE85	18
EDA3	85 80	STA \$80	track 18 for BAM
EDA5	20 E5 ED	JSR \$EDE5	mark dir blocks as allocated
EDA8	A9 00	LDA #\$00	
EDAA	8D F9 02	STA \$02F9	
EDAD	20 FF EE	JSR \$EEFF	write BAM back to disk
EDB0	4C 94 C1	JMP \$C194	done, prepare disk status
*****			
EDB3	C8	INY	
EDB4	B1 94	LDA (\$94),Y	save track
EDB6	48	PHA	
EDB7	C8	INY	
EDB8	B1 94	LDA (\$94),Y	and sector
EDBA	48	PHA	
EDBB	A0 13	LDA #\$13	pointer to side-sector block
EDBD	B1 94	LDA (\$94),Y	
EDBF	F0 0A	BEQ \$EDCB	no track following?
EDC1	85 80	STA \$80	track and
EDC3	C8	INY	
EDC4	B1 94	LDA (\$94),Y	
EDC6	85 81	STA \$81	sector of 1st side-sector block
EDC8	20 E5 ED	JSR \$EDE5	mark side-sector blocks as
EDCB	68	PLA	allocated
EDCC	85 81	STA \$81	
EDCE	68	PLA	get track and sector back
EDCF	85 80	STA \$80	
EDD1	20 E5 ED	JSR \$EDE5	mark blocks of file as allocated
EDD4	20 04 C6	JSR \$C604	read next entry in directory

# Anatomy of the 1541 Disk Drive

EDD7	F0 C3	BEQ \$ED9C	end of directory?
EDD9	A0 00	LDY #\$00	
EDDB	B1 94	LDA (\$94),Y	file type
EDDD	30 D4	BMI \$EDB3	bit 7 set, file closed?
EDDF	20 B6 C8	JSR \$C8B6	file type to zero and write BAM
EDE2	4C D4 ED	JMP \$EDD4	

*****			allocate file blocks in BAM
EDE5	20 5F D5	JSR \$D55F	check track and sector number
EDE8	20 90 EF	JSR \$EF90	allocate block in BAM
EDEB	20 75 D4	JSR \$D475	read next block
EDEE	A9 00	LDA #\$00	
EDF0	20 C8 D4	JSR \$D4C8	buffer pointer zero
EDF3	20 37 D1	JSR \$D137	get byte from buffer
EDF6	85 80	STA \$80	track
EDF8	20 37 D1	JSR \$D137	get byte from buffer
EDFB	85 81	STA \$81	sector
EDFD	A5 80	LDA \$80	another block?
EDFF	D0 03	BNE \$EE04	yes
EE01	4C 27 D2	JMP \$D227	close channel
EE04	20 90 EF	JSR \$EF90	allocate block in BAM
EE07	20 4D D4	JSR \$D44D	read next block
EE0A	4C EE ED	JMP \$EDEC	continue

*****			N command, 'header'
EE0D	20 12 C3	JSR \$C312	get drive number
EE10	A5 E2	LDA \$E2	drive number
EE12	10 05	BPL \$EE19	not clear?
EE14	A9 33	LDA #\$33	
EE16	4C C8 C1	JMP \$C1C8	33, 'syntax error'
EE19	29 01	AND #\$01	
EE1B	85 7F	STA \$7F	drive number
EE1D	20 00 C1	JSR \$C100	turn LED on
EE20	A5 7F	LDA \$7F	drive number
EE22	0A	ASL A	times 2
EE23	AA	TAX	
EE24	AC 7B 02	LDY \$027B	comma position
EE27	CC 74 02	CPY \$0274	compare with end name
EE2A	F0 1A	BEQ \$EE46	format without ID
EE2C	B9 00 02	LDA \$0200,Y	first character of ID
EE2F	95 12	STA \$12,X	save
EE31	B9 01 02	LDA \$0201,Y	second character
EE34	95 13	STA \$13,X	
EE36	20 07 D3	JSR \$D307	close all channels
EE39	A9 01	LDA #\$01	
EE3B	85 80	STA \$80	track 1
EE3D	20 C6 C8	JSR \$C8C6	format disk
EE40	20 05 F0	JSR \$F005	erase buffer
EE43	4C 56 EE	JMP \$EE56	continue as below
EE46	20 42 D0	JSR \$D042	load BAM
EE49	A6 7F	LDX \$7F	drive number
EE4B	BD 01 01	LDA \$0101,X	
EE4E	CD D5 FE	CMP \$FED5	'A', marker for 1541 format

# Anatomy of the 1541 Disk Drive

EE51	F0 03	BEQ \$EE56	ok
EE53	4C 72 D5	JMP \$D572	73, 'cbm dos v2.6 1541'
EE56	20 B7 EE	JSR \$EEB7	create BAM
EE59	A5 F9	LDA SF9	buffer number
EE5B	A8	TAY	
EE5C	0A	ASL A	
EE5D	AA	TAX	
EE5E	AD 88 FE	LDA \$FE88	\$90, start of disk name
EE61	95 99	STA \$99,X	buffer pointer to name
EE63	AE 7A 02	LDX \$027A	
EE66	A9 1B	LDA #\$1B	27
EE68	20 6E C6	JSR \$C66E	write filenames in buffer
EE6B	A0 12	LDY #\$12	position 18
EE6D	A6 7F	LDX \$7F	drive number
EE6F	AD D5 FE	LDA \$FED5	'A', 1541 format
EE72	9D 01 01	STA \$0101,X	
EE75	8A	TXA	
EE76	0A	ASL A	times 2
EE77	AA	TAX	
EE78	B5 12	LDA \$12,X	ID, first character
EE7A	91 94	STA (\$94),Y	in buffer
EE7C	C8	INY	
EE7D	B5 13	LDA \$13,X	and second character
EE7F	91 94	STA (\$94),Y	in buffer
EE81	C8	INY	
EE82	C8	INY	
EE83	A9 32	LDA #\$32	'2'
EE85	91 94	STA (\$94),Y	in buffer
EE87	C8	INY	
EE88	AD D5 FE	LDA \$FED5	'A' 1541 format
EE8B	91 94	STA (\$94),Y	in buffer
EE8D	A0 02	LDY #\$02	
EE8F	91 6D	STA (\$6D),Y	and at position 2
EE91	AD 85 FE	LDA \$FE85	18
EE94	85 80	STA \$80	track number
EE96	20 93 EF	JSR \$EF93	mark block as allocated
EE99	A9 01	LDA #\$01	1
EE9B	85 81	STA \$81	sector number
EE9D	20 93 EF	JSR \$EF93	mark block as allocated
EEA0	20 FF EF	JSR \$EEFF	write BAM
EEA3	20 05 F0	JSR \$F005	pointer \$6D/\$6E to buffer, erase
EEA6	A0 01	LDY #\$01	buffer
EEA8	A9 FF	LDA #\$FF	
EEAA	9A 6D	STA (\$6D),Y	track following is zero
EEAC	20 64 D4	JSR \$D464	write BAM
EEAF	C6 81	DEC \$81	decrement sector number, 0
EEB1	20 60 D4	JSR \$D460	read block
EEB4	4C 94 C1	JMP \$C194	prepare disk status
*****			
EEB7	20 D1 F0	JSR \$F0D1	create RAM
EEBA	A0 00	LDY #\$00	
EEBC	A9 12	LDA #\$12	18
EEBE	91 6D	STA (\$6D),Y	pointer to directory track

# Anatomy of the 1541 Disk Drive

EEC0	C8	INY	
EEC1	98	TYA	1
EEC2	91 6D	STA (\$6D),Y	pointer to directory sector
EEC4	C8	INY	
EEC5	C8	INY	
EEC6	C8	INY	
EEC7	A9 00	LDA #\$00	
EEC9	85 6F	STA \$6F	
EECB	85 70	STA \$70	3 bytes = 24 bits for sectors
EECD	85 71	STA \$71	
EECF	98	TYA	byte position
EED0	4A	LSR A	
EED1	4A	LSR A	divided by 4 = track number
EED2	20 4B F2	JSR \$F24B	get number of sectors
EED5	91 6D	STA (\$6D),Y	and in BAM
EED7	C8	INY	
EED8	AA	TAX	
EED9	38	SEC	
EEDA	26 6F	ROL \$6F	
EEDC	26 70	ROL \$70	create bit model
EEDF	26 71	ROL \$71	
EEE0	CA	DEX	
EEE1	D0 F6	BNE \$EED9	
EEE3	B5 6F	LDA \$6F,X	3 bytes
EEE5	91 6D	STA (\$6D),Y	the BAM in buffer
EEE7	C8	INY	
EEE8	E8	INX	
EEE9	E0 03	CPX #\$03	
EEEB	90 F6	BCC \$EEE3	
EEED	C0 90	CPY #\$90	position 144?
EEEF	90 D6	BCC \$EEC7	no, next track
EEF1	4C 75 D0	JMP \$D075	calculate number of free blocks
*****			write BAM if needed
EEF4	20 93 DF	JSR \$DF93	get buffer number
EEF7	AA	TAX	
EEF8	BD 5B 02	LDA \$025B,X	command for disk controller
EEFB	29 01	AND #\$01	
EEFD	85 7F	STA \$7F	isolate drive number
EEFF	A4 7F	LDY \$7F	
EF01	B9 51 02	LDA \$0251,Y	BAM-changed flag set?
EF04	D0 01	BNE \$EF07	yes
EF06	60	RTS	
EF07	A9 00	LDA #\$00	
EF09	99 51 02	STA \$0251,Y	reset BAM-changed flag
EF0C	20 3A EF	JSR \$EF3A	set buffer pointer for BAM
EF0F	A5 7F	LDA \$7F	drive number
EF11	0A	ASL A	times 2
EF12	48	PHA	
EF13	20 A5 F0	JSR \$F0A5	verify RAM entry
EF16	68	PLA	
EF17	18	CLC	
EF18	69 01	ADC #\$01	increment track number
EF1A	20 A5 F0	JSR \$F0A5	verify BAM entry

# Anatomy of the 1541 Disk Drive

EF1D	A5 80	LDA \$80	track
EF1F	48	PHA	
EF20	A9 01	LDA #\$01	
EF22	85 80	STA \$80	track 1
EF24	0A	ASL A	
EF25	0A	ASL A	times 4
EF26	85 6D	STA \$6D	
EF28	20 20 F2	JSR \$F220	verify BAM
EF2B	E6 80	INC \$80	increment track number
EF2D	A5 80	LDA \$80	
EF2F	CD D7 FE	CMP \$FED7	and compare with max val + 1 = 36
EF32	90 F0	RCC \$EF24	ok, next track
EF34	68	PLA	
EF35	85 80	STA \$80	get track number back
EF37	4C 8A D5	JMP \$D58A	write BAM to disk
*****			set buffer pointer for BAM
EF3A	20 0F F1	JSR \$F10F	get 6 for drive 0
EF3D	AA	TAX	
EF3E	20 DF F0	JSR \$F0DF	allocate buffer
EF41	A6 F9	LDX \$F9	buffer number
EF43	BD E0 FE	LDA \$FEE0,X	buffer address, hi byte
EF46	85 6E	STA \$6E	
EF4A	A9 00	LDA #\$00	lo byte
EF48	85 6D	STA \$6D	pointer to \$6D/\$6E
EF4C	60	RTS	
*****			get # of free blocks for dir
EF4D	A6 7F	LDX \$7F	drive number
EF4F	BD FA 02	LDA \$02FA,X	number of blocks, lo
EF52	8D 72 02	STA \$0272	
EF55	BD FC 02	LDA \$02FC,X	number of blocks, hi
EF58	8D 73 02	STA \$0273	in buffer for directory
EF5B	60	RTS	
*****			mark block as free
EF5C	20 F1 EF	JSR \$EFF1	set buffer pointer
EF5F	20 CF EF	JSR \$EFCF	erase bit for sector in BAM
EF62	38	SEC	
EF63	D0 22	BNF \$EF87	block already free, then done
EF65	B1 6D	LDA (\$6D),Y	bit model of BAM
EF67	1D F9 EF	ORA \$EFE9	set bit X, marker for free
EF6A	91 6D	STA (\$6D),Y	
EF6C	20 88 EF	JSR \$EF88	set flag for BAM changed
EF6F	A4 6F	LDY \$6F	
EF71	18	CLC	
EF72	B1 6D	LDY (\$6D),Y	
EF74	69 01	ADC #\$01	increment # of free blocks/track
EF76	91 6D	STA (\$6D),Y	
EF78	A5 80	LDA \$80	track
EF7A	CD 85 FE	CMP \$FE85	equal to 18?
EF7D	F0 3B	BEQ \$EFBA	then skip
EF7F	FE FA 02	INC \$02FA,X	inc # of free blocks in disk
EF82	D0 03	BNE \$EF87	
EF84	FE FC 02	INC \$02FC,X	increment number of blocks hi

# Anatomy of the 1541 Disk Drive

```

EF87      60          RTS

***** set flag for 'BAM changed'
EF88      A6 7F      LDX $7F      drive number
EF8A      A9 01      LDA #$01
EF8C      9D 51 02   STA $0251,X   flag = 1
EF8F      60          RTS

***** mark block as allocated
EF90      20 F1 EF   JSR $EFF1     set buffer pointer
EF93      20 CF EF   JSR $EFCF     erase bit for sector in BAM
EF96      F0 36      BEQ $EFCE     already allocated, then done
EF98      B1 6D      LDA ($6D),Y
EF9A      5D E9 EF   EOR $EFE9,X   erase bit for block
EF9D      91 6D      STA ($6D),Y
EF9F      20 88 EF   JSR $EF88     set flag for BAM changed
EFA2      A4 6F      LDA $6F
EFA4      B1 6D      LDA ($6D),Y
EFA6      38          SEC
EFA7      E9 01      SBC #$01      decrement # of blocks per track
EFA9      91 6D      STA ($6D),Y
EFAB      A5 80      LDA $80      track
EFAD      CD 85 FE   CMP $FE85     18?
EFB0      F0 0B      BEQ $EFBD
EFB2      BD FA 02   LDA $02FA,X   number of free blocks lo
EFB5      D0 03      BNE $EFBA
EFB7      DE FC 02   DEC $02FC,X   decrement number of free blocks
EFBA      DE FA 02   DEC $02FA,X
EFBD      BD FC 02   LDA $02FC,X   number of free blocks hi
EFC0      D0 0C      BNE $EFCE     more than 255 blocks free?
EFC2      BD FA 02   LDA $02FA,X   free blocks lo
EFC5      C9 03      CMP #$03
EFC7      B0 05      BCS $EFCE     smaller than 3?
EFC9      A9 72      LDA #$72
EFCB      20 C7 E6   JSR $E6C7     72, 'disk full'
EFCE      60          RTS

***** erase bit for sector in RAM entry
EFCF      20 11 F0   JSR $F011     find RAM field for this track
EFD2      98          TYA
EFD3      85 6F      STA $6F
EFD5      A5 81      LDA $81      sector
EFD7      4A          LSR A
EFD8      4A          LSR A      divide by 8
EFD9      4A          LSR A
EFDA      38          SEC
EFDB      65 6F      ADC $6F
EFDD      A8          TAY
EFDE      A5 81      LDA $81      byte number in RAM entry
EFE0      29 07      AND #$07     sector number
EFE2      AA          TAX
EFE3      B1 6D      LDA ($6D),Y   bit number in BAM entry
EFE5      3D E9 EF   AND $EFE9,X   byte in BAM
EFEB      60          RTS          erase bit for corresponding
                                   sector

```

## Anatomy of the 1541 Disk Drive

```

***** powers of 2
EFE9 01 02 04 08 10 20 40 80

***** write RAM after change
EFF1 A9 FF LDA #$FF
EFF3 2C F9 02 BIT $02F9
EFF6 F0 0C BEQ $F004
EFF8 10 0A BPL $F004
EFFA 70 08 BVS $F004
EFFC A9 00 LDA #$00
EF FE 8D F9 02 STA $02F9 reset flag
F001 4C 8A D5 JMP $D58A write block
F004 60 RTS

***** erase BAM buffer
F005 20 3A EF JSR $EF3A pointer $6D/$6E to BAM buffer
F008 A0 00 LDY #$00
F00A 98 TYA
F00B 91 6D STA ($6D),Y erase BAM buffer
F00D C8 INY
F00E D0 FB BNE $F00B
F010 60 RTS

*****
F011 A5 6F LDA $6F
F013 48 PHA
F014 A5 70 LDA $70
F016 48 PHA
F017 A6 7F LDX $7F drive number
F019 B5 FF LDA $FF,X
F01B F0 05 BEQ $F022 drive zero?
F01D A9 74 LDA #$74
F01F 20 48 E6 JSR $F648 'drive not ready'
F022 20 0F F1 JSR $F10F get buffer number for BAM
F025 85 6F STA $6F
F027 8A TXA
F028 0A ASL A
F029 85 70 STA $70
F02B AA TAX
F02C A5 80 LDA $80 track
F02F DD 9D 02 CMP $029D,X
F031 F0 0B BEQ $F03E
F033 E8 INX
F034 86 70 STX $70
F036 DD 9D 02 CMP $029D,X
F039 F0 03 BEQ $F03E
F03B 20 5B F0 JSR $F05B
F03E A5 70 LDA $70
F040 A6 7F LDX $7F drive number
F042 9D 9B 02 STA $029B,X
F045 0A ASL A
F046 0A ASL A times 4
F047 18 CLC
F048 69 A1 ADC #$A1
F04A 85 6D STA $6D

```

# Anatomy of the 1541 Disk Drive

```
F04C  A9 02      LDA #$02
F04E  69 00      ADC #$00
F050  85 6E      STA $6E
F052  A0 00      LDY #$00
F054  58         PLA
F055  85 70      STA $70
F057  68         PLA
F058  85 6F      STA $6F
F05A  60         RTS
```

\*\*\*\*\*

```
F05B  A6 6F      LDX $6F
F05D  20 DF F0    JSR $F0DF
F060  A5 7F      LDA $7F      drive number
F062  AA         TAX
F063  0A         ASL A
F064  1D 9B 02    ORA $029B,X
F067  49 01      EOR #$01
F069  29 03      AND #$03
F06B  85 70      STA $70
F06D  20 A5 F0    JSR $F0A5
F070  A5 F9      LDA $F9      buffer number
F072  0A         ASL A
F073  AA         TAX
F074  A5 80      LDA $80      track
F076  0A         ASL A
F077  0A         ASL A      times 4
F078  95 99      STA $99,X    equal pointer in RAM field
F07A  A5 70      LDA $70
F07C  0A         ASL A
F07D  0A         ASL A
F07E  A8         TAY
F07F  A1 99      LDA ($99,X)
F081  99 A1 02    STA $02A1,X
F084  A9 00      LDA #$00
F086  81 99      STA ($99,X)  zero in buffer
F088  F6 99      INC $99,X    increment buffer pointer
F08A  C8         INY
F08B  98         TYA
F08C  29 03      AND #$03
F08E  D0 EF      BNE $F07F
F090  A6 70      LDX $70
F092  A5 80      LDA $80      track
F094  9D 9D 02    STA $029D,X
F097  AD F9 02    LDA $02F9
F09A  D0 03      BNE $F09F
F09C  4C 8A D5    JMP $D58A    write block

F09F  09 80      ORA #$80
F0A1  8D F9 02    STA $02F9
F0A4  60         RTS

F0A5  A8         TAY
F0A6  B9 9D 02    LDA $029D,Y
F0A9  F0 25      BEQ $F0D0
```



# Anatomy of the 1541 Disk Drive

F0AB	48		PHA	
F0AC	A9	00	LDA #\$00	
F0AE	99	9D 02	STA \$029D,Y	
F0B1	A5	F9	LDA \$F9	buffer number
F0B3	0A		ASL A	times 2
F0B4	AA		TAX	
F0B5	68		PLA	
F0B6	0A		ASL A	
F0B7	0A		ASL A	
F0BB	95	99	STA \$99,X	
F0BA	98		TYA	
F0BB	0A		ASL A	
F0BC	0A		ASL A	
F0BD	AB		TAY	
F0BE	B9	A1 02	LDA \$02A1,Y	
F0C1	81	99	STA (\$99,X)	write in buffer
F0C3	A9	00	LDA #\$00	
F0C5	99	A1 02	STA \$02A1,Y	
F0C8	F6	99	INC \$99,X	increment buffer pointer
FOCA	C8		INY	
F0CB	9B		TYA	
F0CC	29	03	AND #\$03	
F0CE	D0	EE	BNE \$FOBE	
F0D0	60		RTS	
F0D1	A5	7F	LDA \$7F	drive number
F0D3	0A		ASL A	
F0D4	AA		TAX	
F0D5	A9	00	LDA #\$00	
F0D7	9D	9D 02	STA \$029D,X	
F0DA	E8		INX	
F0DB	9D	9D 02	STA \$029D,X	
F0DE	60		RTS	
F0DF	B5	A7	LDA \$A7,X	
F0E1	C9	FF	CMP #\$FF	
F0E3	D0	25	BNE \$F10A	
F0E5	8A		TXA	
F0F6	48		PHA	
F0E7	20	8E D2	JSR \$D28E	
F0EA	AA		TAX	
F0EB	10	05	RPL \$F0F2	
F0ED	A9	70	LDA #\$70	
F0EF	20	C8 C1	JSR \$C1C8	70, 'no channel'
F0F2	86	F9	STX \$F9	
F0F4	68		PLA	
F0F5	A8		TAY	
F0F6	8A		TXA	
F0F7	09	80	ORA #\$80	
F0F9	99	A7 00	STA \$00A7,Y	
F0FC	0A		ASL A	
F0FD	AA		TAX	
F0FE	AD	85 FE	LDA \$FE85	18, directory track
F101	95	06	STA \$06,X	save
F103	A9	00	LDA #\$00	0

## Anatomy of the 1541 Disk Drive

F105	95 07	STA \$07,X	as sector
F107	4C 86 D5	JMP \$D586	write block
F10A	29 0F	AND #\$0F	
F10C	85 F9	STA \$F9	buffer number
F10E	60	RTS	
***** get buffer number for BAM			
F10F	A9 06	LDA #\$06	
F111	A6 7F	LDX \$7F	drive number
F113	D0 03	BNE \$F118	
F115	18	CLC	
F116	69 07	ADC #\$07	gives 13 for drive 0
F118	60	RTS	
***** buffer number for BAM			
F119	20 0F F1	JSR \$F10F	get buffer number
F11C	AA	TAX	
F11D	60	RTS	
***** find and allocate free block			
F11E	20 3E DE	JSR \$DE3E	get track and sector number
F121	A9 03	LDA #\$03	
F123	85 6F	STA \$6F	counter
F125	A9 01	LDA #\$01	
F127	0D F9 02	ORA \$02F9	
F12A	8D F9 02	STA \$02F9	
F12D	A5 6F	LDA \$6F	save counter
F12F	48	PHA	
F130	20 11 F0	JSR \$F011	find BAM field for this track
F133	68	PLA	
F134	85 6F	STA \$6F	get counter back
F136	B1 6D	LDA (\$6D),Y	number of free blocks in track
F138	D0 39	BNE \$F173	blocks still free?
F13A	A5 80	LDA \$80	track
F13C	CD 85 FE	CMP \$FE85	18, directory track?
F13F	F0 19	BEQ \$F15A	yes, 'disk full'
F141	90 1C	BCC \$F15F	smaller, then next lower track
F143	E6 80	INC \$80	increment track number
F145	A5 80	LDA \$80	
F147	CD D7 FE	CMP \$FED7	36, highest track number plus one
F14A	D0 E1	BNE \$F12D	no, continue searching this track
F14C	AE 85 FE	LDX \$FE85	18, directory track
F14F	CA	DEX	decrement
F150	86 80	STX \$80	save as track number
F152	A9 00	LDA #\$00	
F154	85 81	STA \$81	begin with sector number zero
F156	C6 6F	DEC \$6F	decrement counter
F158	D0 D3	BNE \$F12D	not yet zero, then continue
F15A	A9 72	LDA #\$72	
F15C	20 C8 C1	JSR \$C1C8	72, 'disk full'
F15F	C6 80	DEC \$80	decrement track number
F161	D0 CA	BNE \$F12D	not yet 0, continue in this track
F163	AE 85 FE	LDX \$FE85	18, directory track
F166	E8	INX	increment

# Anatomy of the 1541 Disk Drive

F167	86 80	STX \$80	save as track number
F169	A9 00	LDA #\$00	
F16B	85 81	STA \$81	begin with sector zero
F16D	C6 6F	DEC \$6F	decrement counter
F16F	D0 BC	BNE \$F12D	not yet zero, then continue
F171	F0 E7	BEQ \$F15A	else 'disk full'
F173	A5 81	LDA \$81	sector number
F175	18	CLC	
F176	65 69	ADC \$69	plus step width (10)
F178	85 81	STA \$81	as new number
F17A	A5 80	LDA \$80	track number
F17C	20 4B F2	JSR \$F24B	get maximum sector number
F17F	8D 4E 02	STA \$024E	
F182	8D 4D 02	STA \$024D	and save
F185	C5 81	CMP \$81	greater than selected sector #?
F187	B0 0C	BCS \$F195	yes
F189	38	SEC	else
F18A	A5 81	LDA \$81	sector number
F18C	ED 4E 02	SBC \$024E	minus maximum sector number
F18F	85 81	STA \$81	save as new sector number
F191	F0 02	BFO \$F195	zero?
F193	C6 81	DEC \$81	else decrement sector no. by one
F195	20 FA F1	JSR \$F1FA	check BAM, find free sector
F198	F0 03	BEQ \$F19D	not found?
F19A	4C 90 EF	JMP \$EF90	allocate block in BAM
F19D	A9 00	LDA #\$00	
F19F	85 81	STA \$81	sector zero
F1A1	20 FA F1	JSR \$F1FA	find free sector
F1A4	D0 F4	BNE \$F19A	found?
F1A6	4C F5 F1	JMP \$F1F5	no, 'dir sector'
*****			find free sector and allocate
F1A9	A9 01	LDA #\$01	
F1AB	0D F9 02	ORA \$02F9	
F1B1	A5 86	LDA \$86	
F1B3	48	PHA	
F1B4	49 01	LDA #\$01	track counter
F1B6	85 86	STA \$86	
F1B8	AD 85 FE	LDA \$FE85	18, directory track
F1BB	38	SEC	
F1BC	E5 86	SBC \$86	minus counter
F1BE	85 80	STA \$80	save as track number
F1C0	90 09	BCC \$F1CB	result <= zero?
F1C2	F0 07	BEQ \$F1CB	then try top half of dir
F1C4	20 11 F0	JSR \$F011	find BAM field for this track
F1C7	B1 6D	LDA (\$6D),Y	no. of free blocks in this track
F1C9	D0 1B	BNE \$F1E6	free blocks exist
F1CB	AD 85 FE	LDA \$FE85	18, directory track
F1CF	18	CLC	
F1CF	65 86	ADC \$86	plus counter
F1D1	85 80	STA \$80	save as track number
F1D3	E6 86	INC \$86	increment counter
F1D5	CD D7 FE	CMP \$FED7	36, max track number plus one
F1D8	90 05	BCC \$F1DF	smaller, then ok

## Anatomy of the 1541 Disk Drive

F242	68	PLA	
F243	85 6F	STA \$6F	
F245	60	RTS	
F246	A9 71	LDA #\$71	
F248	20 45 E6	JSR \$E645	71, 'dir error'
***** establish # of sectors per track			
F24B	AE D6 FE	LDX \$FED6	4 different values
F24E	DD D6 FE	CMP \$FED6,X	track number
F251	CA	DEX	
F252	B0 FA	BCS \$F24E	not greater?
F254	BD D1 FE	LDA \$FED1,X	get number of sectors
F257	60	RTS	
F258	60	RTS	
***** initialize disk controller			
F259	A9 6F	LDA #\$6F	bit 4 (write prot) & 7 (SYNC)
F25B	8D 02 1C	STA \$1C02	data direction register port B
F25E	29 F0	AND #\$F0	
F260	8D 00 1C	STA \$1C00	port B, control port
F263	AD 0C 1C	LDA \$1C0C	PCR, control register
F266	29 FE	AND #\$FE	
F268	09 0E	ORA #\$0E	
F26A	09 E0	ORA #\$E0	
F26C	8D 0C 1C	STA \$1C0C	
F26F	A9 41	LDA #\$41	
F271	8D 0B 1C	STA \$1C0B	timer 1 free running, enable
F274	A9 00	LDA #\$00	port A latch
F276	8D 06 1C	STA \$1C06	timer 1 lo latch
F279	A9 3A	LDA #\$3A	
F27B	8D 07 1C	STA \$1C07	timer 1 hi latch
F27E	8D 05 1C	STA \$1C05	timer 1 hi
F281	A9 7F	LDA #\$7F	
F283	8D 0E 1C	STA \$1C0E	erase IRQs
F286	A9 C0	LDA #\$C0	
F288	8D 0D 1C	STA \$1C0D	
F28B	8D 0E 1C	STA \$1C0E	IER, allow interrupts
F28E	A9 FF	LDA #\$FF	
F290	85 3E	STA \$3E	
F292	85 51	STA \$51	track counter for formatting
F294	A9 08	LDA #\$08	8
F296	85 39	STA \$39	constants for block header
F298	A9 07	LDA #\$07	7
F29A	85 47	STA \$47	constants for data block
F29C	A9 05	LDA #\$05	
F29E	85 62	STA \$62	
F2A0	A9 FA	LDA #\$FA	pointer \$62/\$63 to \$FA05
F2A2	85 63	STA \$63	
F2A4	A9 C8	LDA #\$C8	200
F2A6	85 64	STA \$64	
F2A8	A9 04	LDA #\$04	
F2AA	85 5E	STA \$5E	
F2AC	A9 04	LDA #\$04	
F2AE	85 6F	STA \$6F	

# Anatomy of the 1541 Disk Drive

```

***** IRQ routine for disk controller
F2B0  BA      TSX
F2B1  86 49    STX $49      save stack pointer
F2B3  AD 04 1C  LDA $1C04
F2B6  AD 0C 1C  LDA $1C0C    erase interrupt flag from timer
F2B9  09 0E    ORA #$0E
F2BB  8D 0C 1C  STA $1C0C
F2BE  A0 05    LDY #$05
F2C0  B9 00 00  LDA $0000,Y  command for buffer Y?
F2C3  10 2E    BPL $F2F3    no
F2C5  C9 D0    CMP #$D0     exec. code for program in buffer
F2C7  D0 04    BNE $F2CD    no
F2C9  98      TYA
F2CA  4C 70 F3  JMP $F370    execute program in buffer
F2CD  29 01    AND #$01     isolate drive number
F2CF  F0 07    BEQ $F2D8    drive zero?
F2D1  84 3F    STY $3F
F2D3  A9 0F    LDA #$0F     else
F2D5  4C 69 F9  JMP $F969    74, 'drive not ready'

F2D8  AA      TAX
F2D9  85 3D    STA $3D
F2DB  C5 3E    CMP $3E      motor running?
F2DD  F0 0A    BEQ $F2E9    yes
F2DF  20 7E F9  JSR $F97E    turn drive motor on
F2E2  A5 3D    LDA $3D
F2E4  85 3E    STA $3E      set flag
F2E6  4C 9C F9  JMP $F99C    to job loop

F2E9  A5 20    LDA $20
F2EB  30 03    BMI $F2F0     head transport programmed?
F2ED  0A      ASL A
F2EE  10 09    BPL $F2F9
F2F0  4C 9C F9  JMP $F99C    to job loop

F2F3  88      DEY
F2F4  10 CA    BPL $F2C0     check next buffer
F2F6  4C 9C F9  JMP $F99C    to job loop

F2F9  A9 20    LDA #$20
F2FB  85 20    STA $20      program head transport
F2FD  A0 05    LDY #$05
F2FF  84 3F    STY $3F      initialize buffer counter
F301  20 93 F3  JSR $F393    set pointer in buffer
F304  30 1A    BMI $F320     command for buffer?
F306  C6 3F    DEC $3F      decrement counter
F308  10 F7    BPL $F301     check next buffer
F30A  A4 41    LDY $41      buffer number
F30C  20 95 F3  JSR $F395    set pointer in buffer
F30F  A5 42    LDA $42      track difference for last job
F311  85 4A    STA $4A      as counter for head transport
F313  06 4A    ASL $4A
F315  A9 60    LDA #$60     set flag for head transport
F317  85 20    STA $20

```

# Anatomy of the 1541 Disk Drive

F1DA	A9 67	LDA #\$67	
F1DC	20 45 E6	JSR \$E645	67, 'illegal track or sector'
F1DF	20 11 F0	JSR \$F011	find BAM field for this track
F1E2	B1 6D	LDA (\$6D),Y	no. of free blocks in this track
F1E4	F0 D2	BEQ \$F1BB	no more free blocks?
F1E6	68	PLA	
F1E7	85 86	STA \$86	
F1E9	A9 00	LDA #\$00	
F1EB	85 81	STA \$81	sector 0
F1ED	20 FA F1	JSR \$F1FA	find free sector
F1F0	F0 03	BEQ \$F1F5	not found?
F1F2	4C 90 EF	JMP \$EF90	allocate block in BAM
F1F5	A9 71	LDA #\$71	
F1F7	20 45 E6	JSR \$E645	71, 'dir error'
*****		find free sectors in actual track	
F1FA	20 11 F0	JSR \$F011	find BAM field for this track
F1FD	98	TYA	points to # of free blocks
F1FE	48	PHA	
F1FF	20 20 F2	JSR \$F220	verify BAM
F202	A5 80	LDA \$80	track
F204	20 4B F2	JSR \$F24B	get max # of sectors of the track
F207	8D 4E 02	STA \$024E	save
F20A	68	PLA	
F20B	85 6F	STA \$6F	save pointer
F20D	A5 81	LDA \$81	compare sector
F20F	CD 4E 02	CMP \$024E	with maximum number
F212	B0 09	BCS \$F21D	greater than or equal to?
F214	20 D5 EF	JSR \$F2D5	get bit number of sector
F217	D0 06	BNE \$F21F	sector free?
F219	E6 81	INC \$81	increment sector number
F21B	D0 F0	BNE \$F20D	and check if free
F21D	A9 00	LDA #\$00	no sectors free
F21F	60	RTS	
*****		verify no. of free blocks in BAM	
F220	A5 6F	LDA \$6F	
F222	48	PHA	
F223	A9 00	LDA #\$00	
F225	85 6F	STA \$6F	counter to zero
F227	AC 86 FE	LDY \$FE86	4, no. of bytes per track in BAM
F22A	88	DEY	
F22B	A2 07	LDX #\$07	
F22D	B1 6D	LDA (\$6D),Y	
F22F	3D E9 EF	AND \$FE9,X	isolate bit
F232	F0 02	BEQ \$F236	
F234	E6 6F	INC \$6F	increment counter of free sectors
F236	CA	DEX	
F237	10 F4	BPL \$F22D	
F239	88	DEY	
F23A	D0 EF	BNE \$F22B	
F23C	B1 6D	LDA (\$6D),Y	compare with number on diskette
F23E	C5 6F	CMP \$6F	
F240	D0 04	BNE \$F246	not equal, then error

# Anatomy of the 1541 Disk Drive

F319	B1 32	LDA (\$32),Y	get track number from buffer
F31B	85 22	STA \$22	
F31D	4C 9C F9	JMP \$F99C	to job loop
F320	29 01	AND #\$01	isolate drive number
F322	C5 3D	CMP \$3D	equal drive number of last job?
F324	D0 E0	BNE \$F306	no
F326	A5 22	LDA \$22	last track number
F328	F0 12	BEQ \$F33C	equal zero?
F32A	38	SEC	
F32B	F1 32	SBC (\$32),Y	equal track number of this job?
F32D	F0 0D	BEQ \$F33C	yes
F32F	49 FF	EOR #\$FF	
F331	85 42	STA \$42	
F333	E6 42	INC \$42	
F335	A5 3F	LDA \$3F	drive number
F337	85 41	STA \$41	
F339	4C 06 F3	JMP \$F306	
E33C	A2 04	LDX #\$04	
F33E	B1 32	LDA (\$32),Y	track number of the job
F340	85 40	STA \$40	save
F342	DD D6 FE	CMP \$FED6,X	compare with max track number
F345	CA	DEX	
F346	B0 FA	BCS \$F342	greater?
F348	8D D1 FE	LDA \$FED1,X	get # of sectors per track
F34B	85 43	STA \$43	and save
F34D	8A	TXA	
F34E	0A	ASL A	
F34F	0A	ASL A	
F350	0A	ASL A	
F351	0A	ASL A	
F352	0A	ASL A	
F353	85 44	STA \$44	gives 0, 32, 64, 96
F355	AD 00 1C	LDA \$1C00	
F358	29 9F	AND #\$9F	
F35A	05 44	ORA \$44	generate control byte for motor
F35C	8D 00 1C	STA \$1C00	
F35F	A6 3D	LDX \$3D	
F361	A5 45	LDA \$45	command code
F363	C9 40	CMP #\$40	position head?
F365	F0 15	BEQ \$F37C	yes
F367	C9 60	CMP #\$60	command code for prg execution?
F369	F0 03	BEQ \$F36E	yes
F36B	4C B1 F3	JMP \$F3B1	read block header
*****			execute program in buffer
F36E	A5 3F	LDA \$3F	buffer number
F370	18	CLC	
F371	69 03	ADC #\$03	plus 3
F373	85 31	STA \$31	
F375	A9 00	LDA #\$00	equals address of buffer
F377	85 30	STA \$30	
F379	6C 30 00	JMP (\$0030)	execute program in buffer
*****			position head

# Anatomy of the 1541 Disk Drive

F37C	A9 60	LDA #\$60	
F37E	85 20	STA \$20	set flag for head transport
F380	AD 00 1C	LDA \$1C00	
F383	29 FC	AND #\$FC	turn stepper motors on
F385	8D 00 1C	STA \$1C00	
F388	A9 A4	LDA #\$A4	164
F38A	85 4A	STA \$4A	step counter for head transport
F38C	A9 01	LDA #\$01	
F38E	85 22	STA \$22	track number
F390	4C 69 F9	JMP \$F969	ok

*****			initialize pointer in buffer
F393	A4 3F	LDY \$3F	buffer number
F395	B9 00 00	LDA \$0000,Y	command code
F398	48	PHA	save
F399	10 10	BPL \$F3AB	
F39B	29 78	AND #\$78	erase bits 0,1,2. and 7
F39D	85 45	STA \$45	
F39F	98	TYA	buffer number
F3A0	0A	ASL A	times two
F3A1	69 06	ADC #\$06	plus 6
F3A3	85 32	STA \$32	equals pointer to actual buffer
F3A5	98	TYA	buffer number
F3A6	18	CLC	
F3A7	69 03	ADC #\$03	plus 3
F3A9	85 31	STA \$31	equals buffer address hi
F3AB	A0 00	LDY #\$00	
F3AD	84 30	STY \$30	buffer address lo
F3AF	68	PLA	get command code back
F3B0	60	RTS	

*****			read block header, verify ID
F3B1	A2 5A	LDX #\$5A	90
F3B3	86 4B	STX \$4B	counter
F3B5	A2 00	LDX #\$00	
F3B7	A9 52	LDA #\$52	82
F3B9	85 24	STA \$24s	
F3BB	20 56 F5	JSR \$F556	wait for SYNC
F3BE	50 FE	BVC \$F3BE	byte ready?
F3C0	B8	CLV	
F3C1	AD 01 1C	LDA \$1C01	data from read head
F3C4	C5 24	CMP \$24	
F3C6	D0 3F	BNE \$F407	20, 'read error'
F3C8	50 FE	BVC \$F3C8	byte ready?
F3CA	B8	CLV	
F3CB	AD 01 1C	LDA \$1C01	data byte from disk(block header)
F3CE	95 25	STA \$25,X	save 7 bytes
F3D0	E8	INX	
F3D1	E0 07	CPX #\$07	
F3D3	D0 F3	BNE \$F3C8	continue reading
F3D5	20 97 F4	JSR \$F497	
F3D8	A0 04	LDY #\$04	4 bytes plus parity
F3DA	A9 00	LDA #\$00	
F3DC	59 16 00	FOR \$0016,Y	form checksum for header
F3DF	88	DEY	



# Anatomy of the 1541 Disk Drive

F3E0	10	FA	BPL \$F3DC	
F3E2	C9	00	CMP #\$00	parity ok?
F3E4	D0	38	BNE \$F41E	27, 'read error'
F3E6	A6	3E	LDX \$3E	drive number
F3E8	A4	18	LDA \$18	track number of header
F3EA	95	22	STA \$22,X	use as actual track number
F3EC	A5	45	LDA \$45	
F3EE	C9	30	CMP #\$30	code for 'preserve header'
F3F0	F0	1E	BEO \$F410	preserve header
F3F2	A5	3E	LDA \$3E	
F3F4	0A		ASL A	
F3F5	A8		TAY	
F3F6	B9	12 00	LDA \$0012,Y	
F3F9	C5	16	CMP \$16	compare with ID1
F3FB	D0	1E	BNE \$F41B	
F3FD	B9	13 00	LDA \$0013,Y	
F400	C5	17	CMP \$17	compare with ID2
F402	D0	17	BNE \$F41B	<>, then 29, 'disk id mismatch'
F404	4C	23 F4	JMP \$F423	
F407	C6	4B	DEC \$4B	decrement counter for attempts
F409	D0	B0	BNE \$F3BB	and try again
F40B	A9	02	LDA #\$02	else
F40D	20	69 F9	JSR \$F969	20, 'read error'
*****				preserve block header
F410	A5	16	LDA \$16	ID1
F412	85	12	STA \$12	
F414	A5	17	LDA \$17	and ID2
F416	85	13	STA \$13	preserve
F418	A9	01	LDA #\$01	ok
F41A	2C		.BYTE \$2C	
F41B	A9	0B	LDA #\$0B	29, 'disk id mismatch'
F41D	2C		.BYTE \$2C	
F41E	A9	09	LDA #\$09	27, 'write error'
F420	4C	69 F9	JMP \$F969	done
*****				
F423	A9	7F	LDA #\$7F	
F425	85	4C	STA \$4C	
F427	A5	19	LDA \$19	
F429	18		CLC	
F42A	69	02	ADC #\$02	
F42C	C5	43	CMP \$43	
F42E	90	02	BCC \$F432	
F430	E5	43	SBC \$43	
F432	85	4D	STA \$4D	
F434	A2	05	LDX #\$05	
F436	86	3F	STX \$3F	
F438	A2	FF	LDX #\$FF	
F43A	20	93 F3	JSR \$F393	set buffer ptr for disk control.
F43D	10	44	BPL \$F483	
F43F	85	44	STA \$44	
F441	29	01	AND #\$01	
F443	C5	3E	CMP \$3E	

# Anatomy of the 1541 Disk Drive

```

F4A9  20 E6 F7  JSR $F7E6
F4AC  A5 55      LDA $55
F4AE  85 18      STA $18
F4B0  A5 54      LDA $54
F4B2  85 19      STA $19
F4B4  A5 53      LDA $53
F4B6  85 1A      STA $1A
F4B8  20 E6 F7  JSR $F7E6
F4BB  A5 52      LDA $52
F4BD  85 17      STA $17
F4BF  A5 53      LDA $53
F4C1  85 16      STA $16
F4C3  68         PLA
F4C4  85 31      STA $31
F4C6  68         PLA
F4C7  85 30      STA $30
F4C9  60         RTS

```

get pointer \$30/\$31 back

\*\*\*\*\*

```

F4CA  C9 00      CMP #$00      command code for 'read'?
F4CC  F0 03      BEQ $F4D1      yes
F4CE  4C 6E F5   JMP $F56E      continue checking command code

F4D1  20 0A F5   JSR $F50A      find beginning of data block
F4D4  50 FE      BVC $F4D4      byte ready?
F4D6  B8         CLV
F4D7  AD 01 1C   LDA $1C01      get data byte
F4DA  91 30      STA ($30),Y    and write in buffer
F4DC  C8         INY            256 times
F4DD  D0 F5      BNE $F4D4
F4DF  A0 BA      LDY #$BA
F4E1  50 FE      BVC $F4E1      byte ready?
F4E3  B8         CLV
F4E4  AD 01 1C   LDA $1C01      read bytes
F4E7  99 00 01   STA $0100,Y    from $1BA to $1FF
F4EA  C8         INY
F4EB  D0 F4      RNE $F4F1
F4ED  20 E0 F8   JSR $F8E0
F4F0  A5 38      LDA $38
F4F2  C5 47      CMP $47      equal 7, beginning of data block
F4F4  F0 05      BEQ $F4FB      yes
F4F6  A9 04      LDA #$04      22, 'read error'
F4F8  4C 69 F9   JMP $F969      error termination

F4FB  20 E9 F5   JSR $F5E9      calculate parity of data block
F4FE  C5 3A      CMP $3A      agreement?
F500  F0 03      BEQ $F505      yes
F502  A9 05      LDA #$05      23, 'read error'
F504  2C         .BYTE $2C
F505  A9 01      LDA #$01      ok
F507  4C 69 F9   JMP $F969      prepare error message

*****
F50A  20 10 F5   JSR $F510      find start of data block
F50D  4C 56 F5   JMP $F556      read block header
                                wait for SYNC

```

# Anatomy of the 1541 Disk Drive

```

*****
F510  A5 3D      LDA $3D      read block header
F512  0A        ASL A        drive number
F513  AA        TAX
F514  B5 12      LDA $12,X    ID1
F516  85 16      STA $16      save
F518  B5 13      LDA $13,X    ID2
F51A  85 17      STA $17      save
F51C  A0 00      LDY #$00
F51E  B1 32      LDA ($32),Y  get track and
F520  85 18      STA $18
F522  C8        INY
F523  B1 32      LDA ($32),Y  sector number from buffer
F525  85 19      STA $19
F527  A9 00      LDA #$00
F529  45 16      FOR $16
F52B  45 17      EOR $17      calculate parity for block header
F52D  45 18      EOR $18
F52F  45 19      EOR $19
F531  85 1A      STA $1A      and save
F533  20 34 F9   JSR $F934
F536  A2 5A      LDX #$5A      90 attempts
F538  20 56 F5   JSR $F556      wait for SYNC
F53B  A0 00      LDY #$00
F53D  50 FE      BVC $F35D      byte ready?
F53F  B8        CLV
F540  AD 01 1C   LDA $1C01      read data from block header
F543  D9 24 00   CMP $0024,Y    compare with saved data
F546  D0 06      BNE $F54E      not the same, try again
F548  C8        INY
F549  C0 08      CPY #$08      8 bytes read?
F54B  D0 F0      BNE $F53D      no
F54D  60        RTS

F54E  CA        DEX            decrement counter
F54F  D0 E7      BNE $F538      not yet zero?
F551  A9 02      LDA #$02
F553  4C 69 F9   JMP $F969      20, 'read error'

*****
F556  A9 D0      LDA #$D0      wait for SYNC
F558  8D 05 18   STA $1805      208
F55B  A9 03      LDA #$03      start timer
F55D  2C 05 18   BIT $1805      error code
F560  10 F1      BPL $F553      timer run down, then 'read error'
F562  2C 00 1C   BIT $1C00      SYNC signal
F565  30 F6      BMI $F55D      not yet found?
F567  AD 10 1C   LDA $1C01      read byte
F56A  B8        CLV
F56B  A0 00      LDY #$00
F56D  60        RTS

*****
F56E  C9 10      CMP #$10      command code for 'write'

```

# Anatomy of the 1541 Disk Drive

F445	D0 3C	BNE \$F483	
F447	A0 00	LDY #S00	
F449	B1 32	LDA (\$32),Y	
F44B	C5 40	CMP S40	
F44D	D0 34	BNE \$F483	
F44F	A5 45	LDA S45	command code
F451	C9 60	CMP #\$60	
F453	F0 0C	BEO \$F461	
F455	A0 01	LDY #S01	
F457	38	SEC	
F458	B1 32	LDA (\$32),Y	
F45A	E5 4D	SBC S4D	
F45C	10 03	BPL \$F461	
F45E	18	CLC	
F45F	65 43	ADC S43	
F461	C4 4C	CMP S4C	
F463	B0 1E	BCS \$F483	
F465	48	PHA	
F466	A5 45	LDA S45	
F468	F0 14	BEO \$F47E	
F46A	68	PLA	
F46B	C9 09	CMP #S09	
F46D	90 14	BCC \$F483	
F46F	C9 0C	CMP #S0C	
F471	B0 10	BCS \$F483	
F473	85 4C	STA S4C	
F475	A5 3F	LDA S3F	
F477	AA	TAX	
F478	69 03	ADC #S03	
F47A	85 31	STA S31	
F47C	D0 05	BNE \$F483	
F47E	68	PLA	
F47F	C9 06	CMP #S06	
F481	90 F0	BCC \$F473	
F483	C6 3F	DEC S3F	
F485	10 B3	BPL \$F43A	
F487	8A	TXA	
F488	10 03	BPL \$F48D	
F48A	4C 9C F9	JMP \$F99C	to job loop
F48D	86 3F	STX S3F	
F48F	20 93 F3	JSR \$F393	get buffer number
F492	A5 45	LDA S45	command code
F494	4C CA F4	JMP \$F4CA	continue checking
F497	A5 30	LDA S30	
F499	48	PLA	save pointer S30/S31
F49A	A5 31	LDA S31	
F49C	48	PHA	
F49D	A9 24	LDA #S24	
F49F	85 30	STA S30	
F4A1	A9 00	LDA #S00	pointer S30/S31 to S24
F4A3	85 31	STA S31	
F4A5	A9 00	LDA #S00	
F4A7	85 34	STA S34	

# Anatomy of the 1541 Disk Drive

F570	F0 03	BEO \$F575	yes
F572	4C 91 F6	JMP \$F691	continue checking command code
*****			
F575	20 E9 F5	JSR \$F5E9	write data block to disk
F57B	85 3A	STA \$3A	calculate parity for buffer
F57A	AD 00 1C	LDA \$1C00	and save
F57D	29 10	AND #\$10	read port B
F57F	D0 05	BNE \$F586	isolate bit for 'write protect'
F581	A9 08	LDA #\$08	not set, ok
F583	4C 69 F9	JMP \$F969	26, 'write protect'
F586	20 8F F7	JSR \$F78F	
F589	20 10 F5	JSR \$F510	find block header
F58C	A2 09	LDX #\$09	
F58E	50 FE	BVC \$F58E	byte ready?
F590	B8	CLV	
F591	CA	DEX	
F592	D0 FA	BNE \$F58E	
F594	A9 FF	LDA #\$FF	
F596	8D 03 1C	STA \$1C03	port A (write/read head) to
F599	AD 0C 1C	LDA \$1C0C	to output
F59C	29 1F	AND #\$1F	
F59E	09 C0	ORA #\$C0	change PCR to output
F5A0	8D 0C 1C	STA \$1C0C	
F5A3	A9 FF	LDA #\$FF	
F5A5	A2 05	LDX #\$05	
F5A7	8D 01 1C	STA \$1C01	write \$FF to disk 5 times
F5AA	B8	CLV	
F5AB	50 FF	BVC \$F5AB	as SYNC characters
F5AD	B8	CLV	
F5AE	CA	DEX	
F5AF	D0 FA	BNE \$F5AB	
F5B1	A0 BB	LDY #\$BB	
F5B3	B9 00 01	LDA \$0100,Y	bytes \$1BB to \$1FF to disk
F5B6	50 FE	BVC \$F5B6	
F5B8	B8	CLV	
F5B9	8D 01 1C	STA \$1C01	
F5BC	C8	INY	
F5BD	D0 F4	BNE \$F5B3	
F5BF	R1 30	LDA (\$30),Y	write data buffer (256 bytes)
F5C1	50 FE	BVC \$F5C1	
F5C3	B8	CLV	
F5C4	8D 01 1C	STA \$1C01	
F5C7	C8	INY	
F5C8	D0 F5	BNE \$F5BF	
F5CA	50 FE	BVC \$F5CA	byte ready?
F5CC	AD 0C 1C	LDA \$1C0C	
F5CF	09 E0	ORA #\$E0	PCR to input again
F5D1	8D 0C 1C	STA \$1C0C	
F5D4	A9 00	LDA #\$00	
F5D6	8D 03 1C	LDA \$1C03	port A (read/write head) to input
F5D9	20 F2 F5	JSR \$F5F2	
F5DC	A4 3F	LDY \$3F	
F5DE	B9 00 00	LDA \$0000,Y	

# Anatomy of the 1541 Disk Drive

F5E1	49 30	EOR #S30	convert command code 'write'
F5E3	99 00 00	STA \$0000,Y	to 'verify'
F5E6	4C B1 F3	JMP \$F3B1	

\*\*\*\*\* calculate parity for data buffer

F5E9	A9 00	LDA #S00
F5EB	A8	TAY
F5EC	51 30	EOR (\$30),Y
F5EE	C8	INY
F5EF	D0 FB	BNE \$F5EC
F5F1	60	RTS

F5F2	A9 00	LDA #S00
F5F4	85 2E	STA \$2E
F5F6	85 30	STA \$30
F5F8	85 4F	STA \$4F
F5FA	A5 31	LDA \$31
F5FC	85 4E	STA \$4E
F5FE	A9 01	LDA #S01
F600	85 31	STA \$31
F602	85 2F	STA \$2F
F604	A9 BB	LDA #SBB
F606	85 34	STA \$34
F608	85 36	STA \$36
F60A	20 E6 F7	JSP \$F7E6
F60D	A5 52	LDA \$52
F60F	85 38	STA \$38
F611	A4 36	LDY \$36
F613	A5 53	LDA \$53
F615	91 2E	STA (\$2E),Y
F617	C8	INY
F618	A5 54	LDA \$54
F61A	91 2E	STA (\$2E),Y
F61C	C8	INY
F61D	A5 55	LDA \$55
F61F	91 2F	STA (\$2F),Y
F621	C8	INY
F622	84 36	STY \$36
F624	20 E6 F7	JSR \$F7E6
F627	A4 36	LDY \$36
F629	A5 52	LDA \$52
F62B	91 2E	STA (\$2E),Y
F62D	C8	INY
F62E	A5 53	LDA \$53
F630	91 2E	STA (\$2E),Y
F632	C8	INY
F633	F0 0E	BEQ \$F643
F635	A5 54	LDA \$54
F637	91 2E	STA (\$2E),Y
F639	C8	INY
F63A	A5 55	LDA \$55
F63C	91 2E	STA (\$2E),Y
F63E	C8	INY
F63F	84 36	STY \$36
F641	D0 E1	BNE \$F624

# Anatomy of the 1541 Disk Drive

```

F643  A5 54      LDA $54
F645  91 30      STA ($30),Y
F647  C8         INY
F648  A5 55      LDA $55
F64A  91 30      STA ($30),Y
F64C  C8         INY
F64D  84 36      STY $36
F64F  20 E6 F7   JSR $F7E6
F652  A4 36      LDY $36
F654  A5 52      LDA $52
F656  91 30      STA ($30),Y
F658  C8         INY
F659  A5 53      LDA $53
F65B  91 30      STA ($30),Y
F65D  C8         INY
F65E  A5 54      LDA $54
F660  91 30      STA ($30),Y
F662  C8         INY
F663  A5 55      LDA $55
F665  91 30      STA ($30),Y
F667  C8         INY
F668  84 36      STY $36
F66A  C0 BB      CPY #SBB
F66C  90 E1      BCC $F64F
F66E  A9 45      LDA #$45
F670  85 2E      STA $2E
F672  A5 31      LDA $31
F674  85 2F      STA $2F
F676  A0 BA      LDY #SBA
F678  B1 30      LDA ($30),Y
F67A  91 2E      STA ($2E),Y
F67C  88         DEY
F67D  D0 F9      BNE $F678
F67F  B1 30      LDA ($30),Y
F681  91 2E      STA ($2E),Y
F683  A2 BB      LDX #SBB
F685  BD 00 01   LDA $0100,X
F688  91 30      STA ($30),Y
F68A  C8         INY
F68B  F8         INX
F68C  D0 F7      BNE $F685
F68E  86 50      STX $50
F690  60         RTS

```

\*\*\*\*\*

```

F691  C9 20      CMP #$20      command code for 'verify'?
F693  F0 03      BEQ $F698      yes
F695  4C CA F6   JMP $F6CA      continue checking command code

F698  20 E9 F5   JSR $F5E9      calculate parity for data buffer
F69B  85 3A      STA $3A          and save
F69D  20 8F F7   JSR $F78F
F6A0  20 0A F5   JSR $F50A      find start of data block
F6A3  A0 BB      LDY #SBB
F6A5  B9 00 01   LDA $0100,Y    data from buffer

```

# Anatomy of the 1541 Disk Drive

F6A8	50 FE	BVC \$F6A8	byte ready?
F6AA	B8	CLV	
F6AB	4D 01 1C	EOR \$1C01	compare with data from disk
F6AE	D0 15	BNE \$F6C5	not equal, then error
F6B0	C8	INY	
F6B1	D0 F2	BNE \$F6A5	
F6B3	B1 30	LDA (\$30),Y	data from buffer
F6B5	50 FE	BVC \$F6B5	
F6B7	B8	CLV	
F6B8	4D 01 1C	EOR \$1C01	compare with data from disk
F6BB	D0 08	BNE \$F6C5	not equal, then error
F6BD	C8	INY	
F6BE	C0 FD	CPY #\$FD	
F6C0	D0 F1	BNE \$F6B3	
F6C2	4C 18 F4	JMP \$F418	error free termination
F6C5	A9 07	LDA #\$07	
F6C7	4C 69 F9	JMP \$F969	25, 'write error'

\*\*\*\*\*

F6CA	20 10 F5	JSR \$F510	read block header
F6CD	4C 18 F4	JMP \$F418	done

\*\*\*\*\*

F6D0	A9 00	LDA #\$00	
F6D2	85 57	STA \$57	
F6D4	85 5A	STA \$5A	
F6D6	A4 34	LDY \$34	
F6D8	A5 52	LDA \$52	
F6DA	29 F0	AND #\$F0	isolate hi-nibble
F6DC	4A	LSR A	
F6DD	4A	LSR A	and rotate to lower nibble
F6DE	4A	LSR A	
F6DF	4A	LSR A	
F6E0	AA	TAX	as index in table
F6E1	BD 7F F7	LDA \$F77F,X	
F6E4	0A	ASL A	
F6E5	0A	ASL A	times 8
F6E6	0A	ASL A	
F6E7	85 56	STA \$56	
F6E9	A5 52	LDA \$52	
F6EB	29 0F	AND #\$0F	isolate lower nibble
F6ED	AA	TAX	as index in table
F6EE	BD 7F F7	LDA \$F77F,X	
F6F1	6A	ROR A	
F6F2	66 57	ROR \$57	
F6F4	6A	ROR A	
F6F5	66 57	ROR \$57	
F6F7	29 07	AND #\$07	
F6F9	05 56	ORA \$56	
F6FB	91 30	STA (\$30),Y	in buffer
F6FD	C8	INY	increment buffer
F6FE	A5 53	LDA \$53	
F700	29 F0	AND #\$F0	isolate upper nibble
F702	4A	LSR A	



# Anatomy of the 1541 Disk Drive

F703	4A	LSR A	
F704	4A	LSR A	shift to upper nibble
F705	4A	LSR A	
F706	AA	TAX	as index in table
F707	BD 7F F7	LDA \$F77F,X	
F70A	0A	ASL A	
F70B	05 57	ORA \$57	
F70D	85 57	STA \$57	
F70F	A5 53	LDA \$53	
F711	29 0F	AND #\$0F	lower nibble
F713	AA	TAX	as index
F714	BD 7F F7	LDA \$F77F,X	
F717	2A	ROL A	
F718	2A	ROL A	
F719	2A	ROL A	
F71A	2A	ROL A	
F71B	85 58	STA \$58	
F71D	2A	ROL A	
F71E	29 01	AND #\$01	
F720	05 57	ORA \$57	
F722	91 30	STA (\$30),Y	in buffer
F724	C8	INY	increment buffer
F725	A5 54	LDA \$54	
F727	29 F0	AND #\$F0	isolate hi-nibble
F729	4A	LSR A	
F72A	4A	LSR A	
F72B	4A	LSR A	
F72C	4A	LSR A	
F72D	AA	TAX	
F72E	BD 7F F7	LDA \$F77F,X	
F731	18	CLC	
F732	6A	ROR A	
F733	05 58	ORA \$58	
F735	91 30	STA (\$30),Y	in buffer
F737	C8	INY	increment buffer pointer
F738	6A	ROR A	
F739	29 80	AND #\$80	
F73B	85 59	STA \$59	
F73D	A5 54	LDA \$54	
F73F	29 0F	AND #\$0F	lower nibble
F741	AA	TAX	as index
F742	BD 7F F7	LDA \$F77F,X	
F745	0A	ASL A	
F746	0A	ASL A	
F747	29 7C	AND #\$7C	
F749	05 59	ORA \$59	
F74B	85 59	STA \$59	
F74D	A5 55	LDA \$55	
F74F	29 F0	AND #\$F0	isolate hi-nibble
F751	4A	LSR A	
F752	4A	LSR A	shift to lower nibble
F753	4A	LSR A	
F754	4A	LSR A	
F755	AA	TAX	as index in table
F756	BD 7F F7	LDA \$F77F,X	

# Anatomy of the 1541 Disk Drive

```

F759      6A          ROR A
F75A      66 5A      ROR $5A
F75C      6A          ROR A
F75D      66 5A      ROR $5A
F75F      6A          ROR A
F760      66 5A      ROR $5A
F762      29 03      AND #$03
F764      05 59      ORA $59
F766      91 30      STA ($30),Y    in buffer
F768      C8          INY           increment buffer pointer
F769      D0 04      BNE $F76F
F76B      A5 2F      LDA $2F
F76D      85 31      STA $31
F76F      A5 55      LDA $55
F771      29 0F      AND #$0F      lower nibble
F773      AA          TAX           as index
F774      BD 7F F7    LDA $F77F,X
F777      05 5A      ORA $5A
F779      91 30      STA ($30),Y    in buffer
F77B      C8          INY           increment buffer pointer
F77C      84 34      STY $34        and save
F77E      60          RTS

```

\*\*\*\*\*

```

F77F 0A 0B 12 13 0E 0F 16 17
F787 09 19 1A 1B 0D 1D 1E 15

```

\*\*\*\*\*

```

F78F      A9 00      LDA #$00
F791      85 30      STA $30
F793      85 2E      STA $2E
F795      85 36      STA $36
F797      A9 BB      LDA #$BB
F799      85 34      STA $34
F79B      85 50      STA $50
F79D      A5 31      LDA $31
F79F      85 2F      STA $2F
F7A1      A9 01      LDA #$01
F7A3      85 31      STA $31
F7A5      A5 47      LDA $47
F7A7      85 52      STA $52
F7A9      A4 36      LDY $36
F7AB      B1 2E      LDA ($2E),Y
F7AD      85 53      STA $53
F7AF      C8          INY
F7B0      B1 2E      LDA ($2E),Y
F7B2      85 54      STA $54
F7B4      C8          INY
F7B5      B1 2E      LDA ($2E),Y
F7B7      85 55      STA $55
F7B9      C8          INY
F7BA      84 36      STY $36
F7BC      20 D0 F6    JSR $F6D0
F7BF      A4 36      LDY $36
F7C1      B1 2E      LDA ($2E),Y

```

# Anatomy of the 1541 Disk Drive

F7C3	85 52	STA \$52
F7C5	C8	INY
F7C6	F0 11	BEQ \$F7D9
F7C8	B1 2E	LDA (\$2E),Y
F7CA	85 53	STA \$53
F7CC	C8	INY
F7CD	B1 2E	LDA (\$2E),Y
F7CF	85 54	STA \$54
F7D1	C8	INY
F7D2	B1 2E	LDA (\$2E),Y
F7D4	85 55	STA \$55
F7D6	C8	INY
F7D7	D0 E1	BNE \$F7BA
F7D9	A5 3A	LDA \$3A
F7DB	85 53	STA \$53
F7DD	A9 00	LDA #S00
F7DF	85 54	STA \$54
F7E1	85 55	STA \$55
F7E3	4C D0 F6	JMP \$F6D0
F7E6	A4 34	LDY \$34
F7E8	B1 30	LDA (\$30),Y
F7EA	29 F8	AND #SF8
F7EC	4A	LSR A
F7ED	4A	LSR A
F7EE	4A	LSR A
F7EF	85 56	STA \$56
F7F1	B1 30	LDA (\$30),Y
F7F3	29 07	AND #S07
F7F5	0A	ASL A
F7F6	0A	ASL A
F7F7	85 57	STA \$57
F7F9	C8	INY
F7FA	D0 06	BNE \$F802
F7FC	A5 4E	LDA \$4E
F7FE	85 31	STA \$31
F800	A4 4F	LDY \$4F
F802	B1 30	LDA (\$30),Y
F804	29 C0	AND #SC0
F806	2A	ROL A
F807	2A	ROL A
F808	2A	ROL A
F809	05 57	ORA \$57
F80B	85 57	STA \$57
F80D	B1 30	LDA (\$30),Y
F80F	29 3E	AND #S3E
F811	4A	LSR A
F812	85 58	STA \$58
F814	B1 30	LDA (\$30),Y
F816	29 01	AND #S01
F818	0A	ASL A
F819	0A	ASL A
F81A	0A	ASL A
F81B	0A	ASL A
F81C	85 59	STA \$59

F81E	C8	INY
F81F	B1 30	LDA (\$30),Y
F821	29 F0	AND #\$F0
F823	4A	LSR A
F824	4A	LSR A
F825	4A	LSR A
F826	4A	LSR A
F827	05 59	ORA \$59
F829	85 59	STA \$59
F82B	B1 30	LDA (\$30),Y
F82D	29 0F	AND #\$0F
F82F	0A	ASL A
F830	85 5A	STA \$5A
F832	C8	INY
F833	B1 30	LDA (\$30),Y
F835	29 80	AND #\$80
F837	18	CLC
F838	2A	ROL A
F839	2A	ROL A
F83A	29 01	AND #\$01
F83C	05 5A	ORA \$5A
F83E	85 5A	STA \$5A
F840	B1 30	LDA (\$30),Y
F842	29 7C	AND #\$7C
F844	4A	LSR A
F845	4A	LSR A
F846	85 5B	STA \$5B
F848	B1 30	LDA (\$30),Y
F84A	29 03	AND #\$03
F84C	0A	ASL A
F84D	0A	ASL A
F84E	0A	ASL A
F84F	85 5C	STA \$5C
F851	C8	INY
F852	D0 06	BNE \$F85A
F854	A5 4E	LDA \$4E
F856	85 31	STA \$31
F858	A4 4F	LDY \$4F
F85A	B1 30	LDA (\$30),Y
F85C	29 E0	AND #\$E0
F85E	2A	ROL A
F85F	2A	ROL A
F860	2A	ROL A
F861	2A	ROL A
F862	05 5C	ORA \$5C
F864	85 5C	STA \$5C
F866	B1 30	LDA (\$30),Y
F868	29 1F	AND #\$1F
F86A	85 5D	STA \$5D
F86C	C8	INY
F86D	84 34	STY \$34
F86F	A6 56	LDX \$56
F871	BD A0 F8	LDA \$F8A0,X
F874	A6 57	LDX \$57
F876	1D C0 F8	ORA \$F8C0,X

# Anatomy of the 1541 Disk Drive

```

F879      85 52      STA $52
F87B      A6 58      LDX $58
F87D      BD A0 F8    LDA $F8A0,X
F880      A6 59      LDX $59
F882      1D C0 F8    ORA $F8C0,X
F885      85 53      STA $53
F887      A6 5A      LDX $5A
F889      BD A0 F8    LDA $F8A0,X
F88C      A6 5B      LDX $5B
F88E      1D C0 F8    ORA $F8C0,X
F891      85 54      STA $54
F893      A6 5C      LDX $5C
F895      BD A0 F8    LDA $F8A0,X
F898      A6 5D      LDX $5D
F89A      1D C0 F8    ORA $F8C0,X
F89D      85 55      STA $55
F89F      60          RTS

```

\*\*\*\*\*

```

F8A0 FF FF FF FF FF FF FF FF
F8A8 FF 80 00 10 FF C0 40 50
F8B0 FF FF 20 30 FF F0 60 70
F8B8 FF 90 A0 B0 FF D0 E0 FF

F8C0 FF FF FF FF FF FF FF FF
F8C8 FF 08 00 01 FF 0C 04 05
F8D0 FF FF 02 03 FF 0F 06 07
F8D8 FF 09 0A 0B FF 0D 0E FF

```

\*\*\*\*\*

```

F8E0      A9 00      LDA #$00
F8E2      85 34      STA $34
F8E4      85 2E      STA $2E
F8E6      85 36      STA $36
F8E8      A9 01      LDA #$01
F8EA      85 4E      STA $4E
F8EC      A9 BA      LDA #$BA
F8EE      85 4F      STA $4F
F8F0      A5 31      LDA $31
F8F2      85 2F      STA $2F
F8F4      20 E6 F7    JSR $F7E6
F8F7      A5 52      LDA $52
F8F9      85 38      STA $38
F8FB      A4 36      LDY $36
F8FD      A5 53      LDA $53
F8FF      91 2E      STA ($2E),Y
F901      C8          INY
F902      A5 54      LDA $54
F904      91 2E      STA ($2E),Y
F906      C8          INY
F907      A5 55      LDA $55
F909      91 2E      STA ($2E),Y
F90B      C8          INY
F90C      84 36      STY $36
F90E      20 E6 F7    JSR $F7E6

```

F911	A4 36	LDY \$36	
F913	A5 52	LDA \$52	
F915	91 2E	STA (\$2E),Y	
F917	C8	INY	
F918	F0 11	BEQ \$F92B	
F91A	A5 53	LDA \$53	
F91C	91 2E	STA (\$2E),Y	
F91E	C8	INY	
F91F	A5 54	LDA \$54	
F921	91 2E	STA (\$2E),Y	
F923	C8	INY	
F924	A5 55	LDA \$55	
F926	91 2E	STA (\$2E),Y	
F928	C8	INY	
F929	D0 E1	BNE \$F90C	
F92B	A5 53	LDA \$53	
F92D	85 3A	STA \$3A	
F92F	A5 2F	LDA \$2F	
F931	85 31	STA \$31	
F933	60	RTS	
F934	A5 31	LDA \$31	
F936	85 2F	STA \$2F	
F938	A9 00	LDA #\$00	
F93A	85 31	STA \$31	
F93C	A9 24	LDA #\$24	
F93E	85 34	STA \$34	
F940	A5 39	LDA \$39	
F942	85 52	STA \$52	
F944	A5 1A	LDA \$1A	
F946	85 53	STA \$53	
F948	A5 19	LDA \$19	
F94A	85 54	STA \$54	
F94C	A5 18	LDA \$18	
F94E	85 55	STA \$55	
F950	20 D0 F6	JSR \$F6D0	
F953	A5 17	LDA \$17	
F955	85 52	STA \$52	
F957	A5 16	LDA \$16	
F959	85 53	STA \$53	
F95B	A9 00	LDA #\$00	
F95D	85 54	STA \$54	
F95F	85 55	STA \$55	
F961	20 D0 F6	JSR \$F6D0	
F964	A5 2F	LDA \$2F	
F966	85 31	STA \$31	
F968	60	RTS	
F969	A4 3F	LDY \$3F	
F96B	99 00 00	STA \$0000,Y	
F96E	A5 50	LDA \$50	
F970	F0 03	BEQ \$F975	
F972	20 F2 F5	JSR \$F5F2	
F975	20 8F F9	JSR \$F98F	
F978	A6 49	LDX \$49	get stack pointer back

# Anatomy of the 1541 Disk Drive

F97A	9A		TXS	
F97B	4C	BE F2	JMP	\$F2BE
F97E	A9	A0	LDA	#\$A0
F980	85	20	STA	\$20
F982	AD	00 1C	LDA	\$1C00
F985	09	04	ORA	#\$04
F987	8D	00 1C	STA	\$1C00
F98A	A9	3C	LDA	\$3C
F98C	85	48	STA	\$48
F98E	60		RTS	
F98F	A6	3E	LDX	\$3E
F991	A5	20	LDA	\$20
F993	09	10	ORA	#\$10
F995	85	20	STA	\$20
F997	A9	FF	LDA	#\$FF
F999	85	48	STA	\$48
F99B	60		RTS	
F99C	AD	07 1C	LDA	\$1C07
F99F	8D	05 1C	STA	\$1C05
F9A2	AD	00 1C	LDA	\$1C00
F9A5	29	10	AND	#\$10
F9A7	C5	1E	CMP	\$1E
F9A9	85	1E	STA	\$1E
F9AB	F0	04	BEQ	\$F9B1
F9AD	A9	01	LDA	#\$01
F9AF	85	1C	STA	\$1C
F9B1	AD	FE 02	LDA	\$02FE
F9B4	F0	15	BEQ	\$F9CB
F9B6	C9	02	CMP	#\$02
F9BB	D0	07	BNE	\$F9C1
F9BA	A9	00	LDA	#\$00
F9BC	8D	FE 02	STA	\$02FE
F9BF	F0	0A	BEQ	\$F9CB
F9C1	85	4A	STA	\$4A
F9C3	A9	02	LDA	#\$02
F9C5	8D	FE 02	STA	\$02FE
F9C8	4C	2E FA	JMP	\$FA2E
F9CB	A6	3E	LDX	\$3E
F9CD	30	07	BMI	\$F9D6
F9CF	A5	20	LDA	\$20
F9D1	A8		TAY	
F9D2	C9	20	CMP	#\$20
F9D4	D0	03	BNE	\$F9D9
F9D6	4C	BE FA	JMP	\$FABE
F9D9	C6	48	DEC	\$48
F9DB	D0	1D	BNE	\$F9FA
F9DD	98		TYA	
F9DE	10	04	BPL	\$F9E4
F9E0	29	7F	AND	#\$7F
F9E2	85	20	STA	\$20

turn drive motor off

write protect?

# Anatomy of the 1541 Disk Drive

```

F9E4  29 10      AND #$10
F9E6  F0 12      BEQ $F9FA
F9E8  AD 00 1C    LDA $1C00
F9EB  29 FB      AND #$FB      drive motor on
F9ED  8D 00 1C    STA $1C00
F9F0  A9 FF      LDA #$FF
F9F2  85 3E      STA $3E
F9F4  A9 00      LDA #$00
F9F6  85 20      STA $20
F9F8  F0 DC      BEQ $F9D6
F9FA  98         TYA
F9FB  29 40      AND #$40
F9FD  D0 03      BNE $FA02
F9FF  4C BE FA    JMP $FABE

FA02  6C 62 00    JMP ($0062)

FA05  A5 4A      LDA #$4A
FA07  10 05      BPL $FA0E
FA09  49 FF      EOR #$FF
FA0B  18         CLC
FA0C  69 01      ADC #$01
FA0E  C5 64      CMP $64
FA10  B0 0A      BCS $FA1C
FA12  A9 3B      LDA #$3B
FA14  85 62      STA $62
FA16  A9 FA      LDA #$FA      pointer $62/$63 to $FA3B
FA18  85 63      STA $63
FA1A  D0 12      BNE $FA2E
FA1C  E5 5E      SBC $5E
FA1E  E5 5E      SBC $5E
FA20  85 61      STA $61
FA22  A5 5E      LDA $5E
FA24  85 60      STA $60
FA26  A9 7B      LDA #$7B
FA28  85 62      STA $62
FA2A  A9 FA      LDA #$FA      pointer $62/$63 to $FA7B
FA2C  85 63      STA $63
FA2E  A5 4A      LDA $4A      step counter for head transport
FA30  10 31      BPL $FA63
FA32  E6 4A      INC $4A      increment
FA34  AE 00 1C    LDX $1C00
FA37  CA         DEX
FA38  4C 69 FA    JMP $FA69

*****

FA3B  A5 4A      LDA $4A      step counter for head transport
FA3D  D0 EF      BNE $FA2E      not yet zero?
FA3F  A9 4E      LDA #$4E
FA41  85 62      STA $62
FA43  A9 FA      LDA #$FA      pointer $62/$63 to $FA4E
FA45  85 63      STA $63
FA47  A9 05      LDA #$05
FA49  85 60      STA $60      counter to 5
FA4B  4C BE FA    JMP $FABE

```



# Anatomy of the 1541 Disk Drive

\*\*\*\*\*

FA4E	C6 60	DEC \$60	decrement counter
FA50	D0 6C	BNE \$FARE	not yet zero?
FA52	A5 20	LDA \$20	
FA54	29 BF	AND #\$BF	erase bit 6
FA56	85 20	STA \$20	
FA58	A9 05	LDA #\$05	
FA5A	85 62	STA \$62	
FA5C	A9 FA	LDA #\$FA	pointer \$62/\$63 to FA05
FA5E	85 63	STA \$63	
FA60	4C BE FA	JMP \$FABE	

\*\*\*\*\*

FA63	C6 4A	DEC \$4A	step counter for head transport
FA65	AE 00 1C	LDX \$1C00	
FA68	E8	INX	
FA69	8A	TXA	
FA6A	29 03	AND #\$03	
FA6C	85 4B	STA \$4B	
FA6E	AD 00 1C	LDA \$1C00	
FA71	29 FC	AND #\$FC	
FA73	05 4B	ORA \$4B	stepper motor off
FA75	8D 00 1C	STA \$1C00	
FA78	4C BE FA	JMP \$FABE	

\*\*\*\*\*

FA7B	38	SEC	
FA7C	AD 07 1C	LDA \$1C07	
FA7F	E5 5F	SBC \$5F	
FA81	8D 05 1C	STA \$1C05	
FA84	C6 60	DEC \$60	decrement counter
FA86	D0 0C	RNE \$FA94	not yet zero?
FA88	A5 5E	LDA \$5E	
FA8A	85 60	STA \$60	
FA8C	A9 97	STA #\$97	
FA8E	85 62	STA \$62	
FA90	A9 FA	LDA #\$FA	pointer \$62/\$63 to \$FA97
FA92	85 63	STA \$63	
FA94	4C 2E FA	JMP \$FA2E	

\*\*\*\*\*

FA97	C6 61	DEC \$61	
FA99	D0 F9	RNE \$FA94	
FA9B	A9 A5	LDA #\$A5	
FA9D	85 62	STA \$62	
FA9F	A9 FA	LDA #\$FA	pointer \$62/\$63 to \$FAA5
FAA1	85 63	STA \$63	
FAA3	D0 EF	BNE \$FA94	

\*\*\*\*\*

FAA5	AD 07 1C	LDA \$1C07
FAA8	18	CLC
FAA9	65 5F	ADC \$5F
FAAB	8D 05 1C	STA \$1C05

# Anatomy of the 1541 Disk Drive

FAAE	C6 60	DEC \$60	decrement counter
FAB0	D0 E2	BNE \$FA94	not yet zero?
FAB2	A9 4E	LDA #\$4E	
FAB4	85 62	STA \$62	
FAB6	A9 FA	LDA #\$FA	pointer \$62/\$63 to \$FA4E
FAB8	85 63	STA \$63	
FABA	A9 05	LDA #\$05	
FABC	85 60	STA \$60	counter to 5
FABE	AD 0C 1C	LDA \$1C0C	
FAC1	29 FD	AND #\$FD	erase bit 1
FAC3	8D 0C 1C	STA \$1C0C	
FAC6	60	RTS	
***** formatting			
FAC7	A5 51	LDA \$51	track number
FAC9	10 2A	BPL \$FAF5	formatting already in progress
FACB	A6 3D	LDX \$3D	drive number
FACD	A9 60	LDA #\$60	flag for head transport
FACF	95 20	STA \$20,X	set
FAD1	A9 01	LDA #\$01	
FAD3	95 22	STA \$22,X	set destination track
FAD5	85 51	STA \$51	running track # for format
FAD7	A9 A4	LDA #\$A4	164
FAD9	85 4A	STA \$4A	step counter for head transport
FADB	AD 00 1C	LDA \$1C00	
FADE	29 FC	AND #\$FC	stepper motor on
FAE0	8D 00 1C	STA \$1C00	
FAE3	A9 0A	LDA #\$0A	10
FAE5	8D 20 06	STA \$0620	error counter
FAE8	A9 A0	LDA #\$A0	\$621/\$622 = 4000
FAEA	8D 21 06	STA \$0621	initialize track capacity
FAED	A9 0F	LDA #\$0F	4000 < capacity < 2*4000 bytes
FAEF	8D 22 06	STA \$0622	
FAF2	4C 9C F9	JMP \$F99C	back in job loop
FAF5	A0 00	LDY #\$00	
FAF7	D1 32	CMP (\$32),Y	
FAF9	F0 05	BEQ \$FB00	
FAFB	91 32	STA (\$32),Y	
FAFD	4C 9C F9	JMP \$F99C	to job loop
FB00	AD 00 1C	LDA \$1C00	
FB03	29 10	AND #\$10	write protect?
FB05	D0 05	BNE \$FB0C	no
FB07	A9 08	LDA #\$08	
FB09	4C D3 FD	JMP \$FDD3	26, 'write protect on'
FB0C	20 A3 FD	JSR \$FDA3	write \$FF to disk 10240 times
FB0F	20 C3 FD	JSR \$FDC3	code (\$621/\$622) times to disk
FB12	A9 55	LDA #\$55	\$55
FB14	8D 01 1C	STA \$1C01	to write head
FB17	20 C3 FD	JSR \$FDC3	and (\$621/\$622) times to disk
FB1A	20 00 FE	JSR \$FE00	switch to read
FB1D	20 56 F5	JSR \$F556	set timer, find \$FF (SYNC)
FB20	A9 40	LDA \$40	

## Anatomy of the 1541 Disk Drive

FB22	0D 0B 18	ORA \$180B	timer 1 free running
FB25	8D 0B 18	STA \$180B	
FB28	A9 62	LDA #\$62	98 cycles, about 0.1 ms
FB2A	8D 06 18	STA \$1806	
FB2D	A9 00	LDA #\$00	
FB2F	8D 07 18	STA \$1807	
FB32	8D 05 18	STA \$1805	start timer
FB35	A0 00	LDY #\$00	counter to zero
FB37	A2 00	LDX #\$00	
FB39	2C 00 1C	BIT \$1C00	SYNC found?
FB3C	30 FB	BMI \$FB39	no, wait
FB3E	2C 00 1C	BIT \$1C00	SYNC found?
FB41	10 FB	BPL \$FB3E	wait for SYNC
FB43	AD 04 18	LDA \$1804	reset interrupt flag timer
FB46	2C 00 1C	BIT \$1C00	SYNC found?
FB49	10 11	BPL \$FB5C	not SYNC (\$55)?
FB4B	AD 0D 18	LDA \$180D	interrupt flag register
FB4E	0A	ASL A	shift timer flag
FB4F	10 F5	BPL \$FB46	timer not run down yet?
FB51	E8	INX	increment counter
FB52	D0 EF	BNE \$FB43	
FB54	C8	INY	increment hi-byte of counter
FB55	D0 EC	BNE \$FB43	
FB57	A9 02	LDA #\$02	overflow, then error
FB59	4C D3 FD	JMP \$FDD3	20, 'read error'
FB5C	86 71	STX \$71	
FB5E	84 72	STY \$72	
FB60	A2 00	LDX #\$00	
FB62	A0 00	LDY #\$00	counter to zero again
FB64	AD 04 18	LDA \$1804	reset timer 1 interrupt flag
FB67	2C 00 1C	BIT \$1C00	SYNC found?
FB6A	30 11	BMI \$FB7D	yes
FB6C	AD 0D 18	LDA \$180D	interrupt-flag register
FB6F	0A	ASL A	timer flag to bit 7
FB70	10 F5	BPL \$FB67	no, wait until timer run down
FB72	E8	INX	
FB73	D0 EF	BNE \$FB64	increment counter
FB75	C8	INY	
FB76	D0 EC	BNE \$FB64	
FB78	A9 02	LDA #\$02	overflow, then error
FB7A	4C D3 FD	JMP \$FDD3	20, 'read error'
FB7D	38	SEC	
FB7E	8A	TXA	
FB7F	E5 71	SBC \$71	difference between counter
FB81	AA	TAX	
FB82	85 70	STA \$70	
FB84	98	TYA	and value for \$FF-storage
FB85	E5 72	SBC \$72	
FB87	A8	TAY	bring to \$70/\$71
FB88	85 71	STA \$71	
FB8A	10 0B	BPL \$FB97	difference positive?
FB8C	49 FF	EOR #\$FF	
FB8E	A8	TAY	

# Anatomy of the 1541 Disk Drive

FB8F	8A		TXA	
FB90	49	FF	EOR #\$FF	calculate abs. val of difference
FB92	AA		TAX	
FB93	E8		INX	
FB94	D0	01	BNE \$FB97	
FB96	C8		INY	
FB97	98		TYA	
FB98	D0	04	BNE \$FB9E	
FB9A	E0	04	CPX #\$04	difference less than 4 * 0.1 ms
FB9C	90	18	BCC \$FBB6	yes
FB9E	06	70	ASL \$70	
FBA0	26	71	ROL \$71	double difference
FBA2	18		CLC	
FBA3	A5	70	LDA \$70	
FBA5	6D	21 06	ADC \$0621	
FBA8	8D	21 06	STA \$0621	add to 4000
FBAB	A5	71	LDA \$71	
FBAD	6D	22 06	ADC \$0622	
FBBD	8D	22 06	STA \$0622	
FBB3	4C	0C FB	JMP \$FBC0C	repeat until diff < 4 * 0.1 ms
FBB6	A2	00	LDX #\$00	
FBB8	A0	00	LDY #\$00	counter to zero
FBBA	B8		CLV	
FBBB	AD	00 1C	LDA \$1C00	SYNC?
FBBE	10	0E	BPL \$FBCE	no
FBC0	50	59	BVC \$FBBB	byte ready?
FBC2	B8		CLV	
FBC3	E8		INX	
FBC4	D0	F5	BNE \$FBBB	increment counter
FBC6	C8		INY	
FBC7	D0	F2	BNE \$FBBB	
FBC9	A9	03	LDA #\$03	overflow, then error
FBCB	4C	D3 FD	JMP \$FDD3	21, read error
FBCE	8A		TXA	
FBCF	0A		ASL A	double counter
FBD0	8D	25 06	STA \$0625	
FBD3	98		TYA	
FBD4	2A		ROL A	and to \$624/\$625 as track cap.
FBD5	8D	24 06	STA \$0624	
FBD8	A9	BF	LDA #\$BF	
FBDA	2D	0B 18	AND \$180B	
FBD8	8D	0B 18	STA \$180B	
FBE0	A9	66	LDA #\$66	102
FBE2	8D	26 06	STA \$0626	
FBE5	A6	43	LDX \$43	number of sectors in this track
FBE7	A0	00	LDY #\$00	
FBE9	98		TYA	
FBEA	18		CLC	
FBE8	6D	26 06	ADC \$0626	
FBE8	90	01	BCC \$FBE1	
FBF0	C8		INX	
FBF1	C8		INX	
FBF2	CA		DEX	

# Anatomy of the 1541 Disk Drive

FBF3	D0 F5	BNE \$FBEA	calculate # of bytes
FBF5	49 FF	EOR #\$FF	
FBF7	38	SEC	
FBF8	69 00	ADC #\$00	
FBFA	18	CLC	
FBFB	6D 25 06	ADC \$0625	
FBFE	B0 03	BCS \$FC03	
FC00	CE 24 06	DEC \$0624	
FC03	AA	TAX	
FC04	98	TYA	
FC05	49 FF	EOR #\$FF	
FC07	38	SEC	
FC08	69 00	ADC #\$00	
FC0A	18	CLC	
FC0B	6D 24 06	ADC \$0624	result in A/X
FC0E	10 05	BPL \$FC15	
FC10	A9 04	LDA #\$04	
FC12	4C D3 FD	JMP \$FDD3	22, 'read error'
FC15	A8	TAY	
FC16	8A	TXA	
FC17	A2 00	LDX #\$00	
FC19	38	SEC	total divided by number
FC1A	E5 43	SBC \$43	of sectors (\$43)
FC1C	B0 03	BCS \$FC21	
FC1E	88	DEY	
FC1F	30 03	BMI \$FC24	
FC21	E8	INX	
FC22	D0 F5	BNE \$FC19	
FC24	8E 26 06	STX \$0626	compare no. of bytes per interval
FC27	E0 04	CPX #\$04	with minimum value
FC29	B0 05	BCS \$FC30	ok
FC2B	A9 05	LDA #\$05	
FC2D	4C D3 FD	JMP \$FDD3	23, 'read error'
FC30	18	CLC	remainder of division
FC31	65 43	ADC \$43	plus number of sectors
FC33	8D 27 06	STA \$0627	save
FC36	A9 00	LDA #\$00	
FC38	8D 28 06	STA \$0628	counter for sectors
FC3B	A0 00	LDY #\$00	counter lo
FC3D	A6 3D	LDX \$3D	drive number
FC3F	A5 39	LDA \$39	constant 8, marker for header
FC41	99 00 03	STA \$0300,Y	in buffer
FC44	C8	INY	
FC45	C8	INY	
FC46	AD 28 06	LDA \$0628	sector number
FC49	99 00 03	STA \$0300,Y	in buffer
FC4C	C8	INY	
FC4D	A5 51	LDA \$51	track number
FC4F	99 00 03	STA \$0300,Y	in buffer
FC52	C8	INY	
FC53	B5 13	LDA \$13,X	ID 2
FC55	99 00 03	STA \$0300,Y	in buffer
FC58	C8	INY	
FC59	B5 12	LDA \$12,X	ID 1

# Anatomy of the 1541 Disk Drive

FC5B	99 00 03	STA \$0300,Y	in buffer
FC5E	C8	INY	
FC5F	A9 0F	LDA #\$0F	15
FC61	99 00 03	STA \$0300,Y	in buffer
FC64	C8	INY	
FC65	99 00 03	STA \$0300,Y	15 in buffer
FC68	C8	INY	
FC69	A9 00	LDA #\$00	
FC6B	59 FA 02	EOR \$02FA,Y	
FC6E	59 FB 02	EOR \$02FB,Y	
FC71	59 FC 02	EOR \$02FC,Y	generate checksum
FC74	59 FD 02	EOR \$02FD,Y	
FC77	99 F9 02	STA \$02F9,Y	
FC7A	EE 28 06	INC \$0628	increment counter
FC7D	AD 28 06	LDA \$0628	counter
FC80	C5 43	CMP \$43	compare with no. of sectors
FC82	90 BB	BCC \$FC3F	smaller, then continue
FC84	98	TYA	
FC85	48	PHA	
FC86	E8	INX	
FC87	8A	TXA	
FC88	9D 00 05	STA \$0500,X	
FC8B	E8	INX	
FC8C	D0 FA	BNE \$FC88	
FC8E	A9 03	LDA #\$03	buffer pointer to \$300
FC90	85 31	STA \$31	
FC92	20 30 FE	JSR \$FE30	
FC95	68	PLA	
FC96	A8	TAY	
FC97	88	DEY	
FC98	20 E5 FD	JSR \$FDE5	copy buffer data
FC9B	20 F5 FD	JSR \$FDF5	copy data in buffer
FC9E	A9 05	LDA #\$05	
FCA0	85 31	STA \$31	buffer pointer to \$500
FCA2	20 E9 F5	JSR \$F5E9	calculate parity for data buffer
FCA5	85 3A	STA \$3A	and save
FCA7	20 8F F7	JSR \$F78F	
FCAA	A9 00	LDA #\$00	
FCAC	85 32	STA \$32	
FCAE	20 0E FE	JSR \$FE0E	
FCB1	A9 FF	LDA #\$FF	
FCB3	8D 01 1C	STA \$1C01	to write head
FCB6	A2 05	LDX #\$05	write \$FF 5 times
FCB8	50 FE	BVC \$FCB8	byte ready
FCBA	B8	CLV	
FCBB	CA	DEX	
FCBC	D0 FA	BNE \$FCB8	
FCBE	A2 0A	LDX \$0A	10 times
FCC0	A4 32	LDY \$32	buffer pointer
FCC2	50 FE	BVC \$FCC2	byte ready?
FCC4	B8	CLV	
FCC5	B9 00 03	LDA \$0300,Y	data from buffer
FCC8	8D 01 1C	STA \$1C01	write
FCCB	C8	INY	
FCCC	CA	DEX	10 data written?

# Anatomy of the 1541 Disk Drive

FCCD	D0 F3	BNE \$FCC2	
FCCF	A2 09	LDX #\$09	9 times
FCD1	50 FE	BVC \$FCD1	byte ready?
FCD3	B8	CLV	
FCD4	A9 55	LDA #\$55	\$55
FCD6	8D 01 1C	STA \$1C01	write
FCD9	CA	DEX	
FCDA	D0 F5	BNE \$FCD1	9 times?
FCDC	A9 FF	LDA #\$FF	\$FF
FCDE	A2 05	LDX #\$05	5 times
FCE0	50 FE	BVC \$FCE0	byte ready?
FCE2	B8	CLV	
FCE3	8D 01 1C	STA \$1C01	to write head
FCE6	CA	DEX	
FCE7	D0 F7	BNE \$FCE0	
FCE9	A2 BB	LDX #\$BB	
FCEB	50 FE	BVC \$FCEB	
FCED	B8	CLV	
FCEE	BD 00 01	LDA \$0100,X	area \$1BB to \$1FF
FCF1	8D 01 1C	STA \$1C01	save
FCF4	E8	INX	
FCF5	D0 F4	BNE \$FCEB	
FCF7	A0 00	LDY #\$00	
FCF9	50 FE	BVC \$FCF9	byte ready?
FCFB	B8	CLV	
FCFC	B1 30	LDA (\$30),Y	256 bytes of data
FCFE	8D 01 1C	STA \$1C01	write byte to disk
FD01	C8	INY	
FD02	D0 F5	BNE \$FCF9	
FD04	A9 55	LDA #\$55	\$55
FD06	AE 26 06	LDX \$0626	(\$626) times
FD09	50 FE	BVC \$FD09	
FD0B	B8	CLV	
FD0C	8D 01 1C	STA \$1C01	write
FD0F	CA	DEX	
FD10	D0 F7	BNE \$FD09	
FD12	A5 32	LDA \$32	
FD14	18	CLC	
FD15	69 0A	ADC #\$0A	plus 10
FD17	85 32	STA \$32	
FD19	CE 28 06	DEC \$0628	decrement sector number
FD1C	D0 93	BNE \$FCB1	
FD1E	50 FE	BVC \$FD1E	byte ready?
FD20	B8	CLV	
FD21	50 FE	BVC \$FD21	byte ready?
FD23	B8	CLV	
FD24	20 00 FE	JSR \$FE00	switch to reading
FD27	A9 C8	LDA #\$C8	200
FD29	8D 23 06	STA \$0623	
FD2C	A9 00	LDA #\$00	
FD2E	85 30	STA \$30	
FD30	A9 03	LDA #\$03	buffer pointer to \$200
FD32	85 31	STA \$31	
FD34	A5 43	LDA \$43	number of sectors per track
FD36	8D 28 06	STA \$0628	

# Anatomy of the 1541 Disk Drive

FD39	20 56 F5	JSR \$F556	wait for SYNC
FD3C	A2 0A	LDX #\$0A	10 data
FD3E	A0 00	LDY #\$00	
FD40	50 FE	BVC \$FD40	byte ready?
FD42	B8	CLV	
FD43	AD 01 1C	LDA \$1C01	read byte
FD46	D1 30	CMP (\$30),Y	compare with data in buffer
FD48	D0 0E	BNE \$FD58	not equal, error
FD4A	C8	INY	
FD4B	CA	DEX	
FD4C	D0 F2	BNE \$FD40	
FD4E	18	CLC	
FD4F	A5 30	LDA \$30	
FD51	69 0A	ADC #\$0A	increment pointer by 10
FD53	85 30	STA \$30	
FD55	4C 62 FD	JMP \$FD62	
FD58	CE 23 06	DEC \$0623	decrement counter for attempts
FD5B	D0 CF	BNE \$FD2C	not yet zero?
FD5D	A9 06	LDA #\$06	else error
FD5F	4C D3 FD	JMP \$FDD3	24, 'read error'
FD62	20 56 F5	JSR \$F556	wait for SYNC
FD65	A0 BB	LDY #\$BB	
FD67	50 FE	BVC \$FD67	byte ready?
FD69	B8	CLV	
FD6A	AD 01 1C	LDA \$1C01	read byte
FD6D	D9 00 01	CMP \$0100,Y	compare with buffer contents
FD70	D0 E6	BNE \$FD58	not equal, error
FD72	C8	INY	
FD73	D0 F2	BNE \$FD67	next byte
FD75	A2 FC	LDX #\$FC	
FD77	50 FE	BVC \$FD77	byte ready?
FD79	B8	CLV	
FD7A	AD 01 1C	LDA \$1C01	read byte
FD7D	D9 00 05	CMP \$0500,Y	compare with buffer contents
FD80	D0 D6	BNE \$FD58	not equal, then error
FD82	C8	INY	
FD83	CA	DEX	
FD84	D0 F1	BNE \$FD77	next byte
FD86	CE 28 06	DEC \$0628	decrement sector counter
FD89	D0 AE	BNE \$FD39	not yet zero?
FD8B	E6 51	INC \$51	increment track number
FD8D	A5 51	LDA \$51	
FD8F	C9 24	CMP #\$24	compare with 36, highest trk# +1
FD91	B0 03	BCS \$FD96	greater, then formatting done
FD93	4C 9C F9	JMP \$F99C	continue
FD96	A9 FF	LDA #\$FF	
FD98	85 51	STA \$51	track number to \$FF
FD9A	A9 00	LDA #\$00	
FD9C	85 50	STA \$50	
FD9E	A9 01	LDA #\$01	
FDA0	4C 69 F9	JMP \$F969	ok



# Anatomy of the 1541 Disk Drive

```
***** write $FF 10240 times
FDA3  AD 0C 1C  LDA $1C0C
FDA6  29 1F      AND #$1F      switch PCR to writing
FDA8  09 C0      ORA #$C0
FDAA  8D 0C 1C  STA $1C0C
FDAD  A9 FF      LDA #$FF
FDAF  8D 03 1C  STA $1C03      port A(read/write head) to output
FDB2  8D 01 1C  STA $1C01      write $FF to disk
FDB5  A2 28      LDX #$28      40
FDB7  A0 00      LDY #$00
FDB9  50 FE      BVC $FDB9      byte ready?
FDBB  B8          CLV
FDBC  88          DEY
FDBD  D0 FA      BNE $FD89
FDBF  CA          DEX
FDC0  D0 F7      BNE $FD89
FDC2  60          RTS

***** read/write ($621/$622) times
FDC3  AE 21 06  LDX $0621
FDC6  AC 22 06  LDY $0622
FDC9  50 FE      BVC $FDC9      byte ready?
FDCB  B8          CLV
FDCC  CA          DEX
FDCD  D0 FA      BNE $FDC9
FDCF  88          DEY
FDD0  10 F7      BPL $FDC9
FDD2  60          RTS

***** attempt counter for formatting
FDD3  CE 20 06  DEC $0620      decrement number of attempts
FDD6  F0 03      BEQ $FDDB      zero, then error
FDD8  4C 9C F9  JMP $F99C      continue

FDDR  A0 FF      LDY #$FF
FDDD  84 51      STY $51      flag for end of formatting
FDDF  C8          INY
FDE0  84 50      STY $50
FDE2  4C 69 F9  JMP $F969      error termination

*****
FDE5  B9 00 03  LDA $0300,Y
FDE8  99 45 03  STA $0345,Y
FDEB  88          DEY      copy buffer contents
FDEC  D0 F7      BNE $FDE5
FDEE  AD 00 03  LDA $0300
FDF1  8D 45 03  STA $0345
FDF4  60          RTS

*****
FDF5  A0 44      LDY #$44
FDF7  B9 BB 01  LDA $01BB,Y      $1BB to $1FF
FDFA  91 30      STA ($30),Y      write in buffer $30/$31
FDFC  88          DEY
FDFD  10 F8      BPL $FDF7
```

# Anatomy of the 1541 Disk Drive

FDFE 60 RTS

```
***** switch to reading
FE00 AD 0C 1C LDA $1C0C
FE03 09 E0 ORA #$E0 switch PCR to reading
FE05 8D 0C 1C STA $1C0C
FE08 A9 00 LDA #$00
FE0A 8D 03 1C STA $1C03 port A to input
FE0D 60 RTS
```

```
***** write $55 10240 times
FE0E AD 0C 1C LDA $1C0C
FE11 29 1F AND #$1F
FE13 09 C0 ORA #$C0 switch PCR to writing
FE15 8D 0C 1C STA $1C0C
FE18 A9 FF LDA #$FF
FE1A 8D 03 1C STA $1C03 port A to output (write head)
FE1D A9 55 LDA #$55 %01010101
FE1F 8D 01 1C STA $1C01 to port A (write head)
FE22 A2 28 LDY #$28
FE24 A0 00 LDY #$00
FE26 50 FE BVC $FE26 byte ready for write electronics
FE28 B8 CLV
FE29 88 DEY
FE2A D0 FA BNE $FE26 10240 times
FE2C CA DEX
FE2D D0 F7 BNE $FE26
FE2F 60 RTS
```

```
*****
FE30 A9 00 LDA #$00
FE32 85 30 STA $30
FE34 85 2E STA $2E
FE36 85 36 STA $36
FE38 A9 BB LDA #$BB
FE3A 85 34 STA $34
FE3C A5 31 LDA $31
FE3E 85 2F STA $2F
FE40 A9 01 LDA #$01
FE42 85 31 STA $31
FE44 A4 36 LDY $36
FE46 B1 2E LDA ($2E),Y
FE48 85 52 STA $52
FE4A C8 INY
FE4B B1 2E LDA ($2E),Y
FE4D 85 53 STA $53
FE4F C8 INY
FE50 B1 2E LDA ($2E),Y
FE52 85 54 STA $54
FE54 C8 INY
FE55 B1 2E LDA ($2E),Y
FE57 85 55 STA $55
FE59 C8 INY
FE5A F0 08 BEQ $FE64
FE5C 84 36 STY $36
```

## Anatomy of the 1541 Disk Drive

FE5E 02 D0 F6 JSR \$F6D0

FE61 4C 44 FE

FE64 4C D0 F6 JMP \$F6D0

```
***** interrupt routine
FE67 48 PHA
FE68 8A TXA
FE69 48 PHA save registers
FE6A 98 TYA
FE6B 48 PHA
FE6C AD 0D 18 LDA $180D interrupt from serial bus
FE6F 29 02 AND #$02
FE71 F0 03 BEQ $FE76 no
FE73 20 53 E8 JSR $E853 serve serial bus
FE76 AD 0D 1C LDA $1C0D interrupt from timer 1?
FE79 0A ASL A
FE7A 10 03 BPL $FE7F no
FE7C 20 B0 F2 JSR $F2R0 IRQ routine for disk controller
FE7F 68 PLA
FE80 A8 TAY
FE81 68 PLA get register back
FE82 AA TAX
FE83 68 PLA
FE84 40 RTI
```

```
***** constants for disk format
FE85 12 18, track for BAM and directory
FE86 04 start of BAM at position 4
FE87 04 4 bytes in BAM for each track
FE88 90 $90 = 144, end of BAM, disk name
```

```
***** table of command words
FE89 56 49 44 4D 42 55 'V', 'I', 'D', 'M', 'B', 'U'
FE8F 50 26 43 52 53 4E 'P', '&', 'C', 'R', 'S', 'N'
```

```
***** 10-bytes of command addresses
FE95 84 05 C1 F8 1B 5C
FE9F 07 A3 F0 88 23 0D
```

```
***** h1-bytes of command addresses
FEA1 ED D0 C8 CA CC CB
FEA7 E2 E7 C8 CA C8 EE
```

```
***** bytes for syntax check
FEAD 51 DD 1C 9E 1C
```

```
***** file control methods
FEB2 52 57 41 4D 'R', 'W', 'A', 'M'
```

```
***** file types
FEB6 44 53 50 55 4C 'D', 'S', 'P', 'U', 'L'
```

```
***** names of file types
FEBB 44 53 50 55 52 1st letters 'D', 'S', 'P', 'U', 'R'
```

## Anatomy of the 1541 Disk Drive

FF0D 85 23 STA \$23  
FF0F 60 RTS

\*\*\*\*\*

FF10 AA ...  
FFE1 ... AA

\*\*\*\*\*

FFE2 52 53 52 AA  
FFE6 C6 C8 8F F9

\*\*\*\*\*

FFEA 5F CD	UA, U1, \$CD5F
FFEC 97 CD	UB, U2, \$CD97
FFEE 00 05	UC, U3, \$0500
FFF0 03 05	UD, U4, \$0503
FFF2 06 05	UE, U5, \$0506
FFF4 09 05	UF, U6, \$0509
FFF6 0C 05	UG, U7, \$050C
FFF8 0F 05	UH, U8, \$050F
FFFA 01 FF	UI, U9, \$FF01

(NMI vector not used)

\*\*\*\*\*

	hardware vectors
FFFC 0A EA	\$EAA0 RESET and UJ (U:) vector
FFFE 67 FE	\$FE67 IRQ vector

## Anatomy of the 1541 Disk Drive

```

FEC0 45 45 52 53 45      2nd letters 'E', 'E', 'R', 'S', 'E'
FEC5 4C 51 47 52 4C      3rd letters 'L', 'O', 'G', 'R', 'L'

*****
FECA 08 00 00

*****
FECD 3F 7F BF FF      masks for bit command

*****      number of sectors per track
FED1 11 12 13 15      17, 18, 19, 21

*****      constants for disk format
FED5 4A      'A' marker for 1541 format
FED6 04      4 track numbers
FED7 24      36, highest track number + 1
FED8 1F 19 12      31, 25, 18 tracks with change of
                   number of sectors

*****
FEDB 01 FF FF 01 00      control bytes for head position

*****      addresses of buffers
FEE0 03 04 05 06 07      high bytes

*****
FEE5 07 0E

*****      for UI command
FEE7 6C 65 00 JMP ($0065)

*****      for diagnostic routine
FEEA 8D 00 1C STA $1C00      turn LED on
FEED 8D 02 1C STA $1C02      port to output
FEF0 4C 7D EA JMP $EA7D      back to diagnostic routine

*****      delay loop for serial bus
FEF3 8A TXA
FEF4 A2 05 LDX #$05
FEF6 CA DEX      about 40 microseconds
FEF7 D0 FD BNE $FEF6
FEF9 AA TAX
FEFA 60 RTS

*****      data output to serial bus
FEFB 20 AE E9 JSR $E9AE      CLOCK OUT hi
FEFE 4C 9C E9 JMP $E99C      DATA OUT lo

*****      UI vector
FF01 AD 02 02 LDA $0202
FF04 C9 2D CMP #$2D
FF06 F0 05 BEQ $FF0D
FF08 38 SEC
FF09 E9 2B SBC #$2B
FF0B D0 DA BNE $FEE7      indirect jump over ($65)

```

# Anatomy of the 1541 Disk Drive

```
FF0D 85 23      STA $23
FF0F 60         RTS
```

\*\*\*\*\*

```
FF10 AA ...
FFE1 ... AA
```

\*\*\*\*\*

```
FFE2 52 53 52 AA
FFE6 C6 C8 8F F9
```

\*\*\*\*\*

	USER vectors
FFEA 5F CD	UA, U1, \$CD5F
FFEC 97 CD	UB, U2, \$CD97
FFEE 00 05	UC, U3, \$0500
FFF0 03 05	UD, U4, \$0503
FFF2 06 05	UE, U5, \$0506
FFF4 09 05	UF, U6, \$0509
FFF6 0C 05	UG, U7, \$050C
FFF8 0F 05	UH, U8, \$050F
FFFA 01 FF	UI, U9, \$FF01
	(NMI vector not used)

\*\*\*\*\*

	hardware vectors
FFFC 0A EA	\$EAA0 RESET and UJ (U:) vector
FFFE 67 FE	\$FE67 IRQ vector

## Chapter 4: Programs and Tips for the 1541 Disk Drive

## 4.1 Utility Programs

## 4.1.1 Displaying all File Parameters

The directory contains several important pieces of information about each file. Some information is not kept in the directory, such as the starting address of a program.

These and other file parameters can be easily found and displayed by the following program. The number and kind of file parameters are naturally dependent on the file type. A relative file, for instance, has no starting address. The following table presents the parameters displayed by this program.

PARAMETER	FILE TYPE				
	DEL	SEO	PRG	USR	REL
File closed?	X	X	X	X	X
File protected?	X	X	X	X	X
Allocated blocks	X	X	X	X	X
Side-sector blocks					X
Data blocks					X
Records					X
Start address			X		
Free blocks, disk	X	X	X	X	X
Allocated bl. disk	X	X	X	X	X

This program is documented in detail so that the serious programmer can get a good overview of the file parameters. In addition, the variables used by the program are explained.

Variables used in the program:

## Numerical Variables

- T - Track of the actual block of the file entry in the directory
- S - Sector of the actual block of the file entry in the directory
- FL - Flag, set if the file name read from the diskette does not agree with the searched-for file
- TY - File type of the given file (byte 0 of the entry)

## Anatomy of the 1541 Disk Drive

FT - nybble of the file type (bits 0-3), contains the actual file type  
LB - Low byte of the starting address  
HB - High byte of the starting address  
BL - Number of allocated blocks in the file  
RL - Record length of a relative file  
DT - Track of the first data block of a program file, which contains the starting address  
DS - Sector of the first data block of a program file  
SA - Starting address of a program file  
BF - Number of free blocks on a disk  
BA - Number of allocated blocks on a disk  
BS - Number of side-sector blocks in a relative file  
RC - Number of records in a relative file

### String Variables

---

F\$ - Name of the file to search for  
F\$ - Contains the actual file name from the directory  
F\$ - File type  
CLS - Indicates whether the file is closed or not (contains "YES" or "NO")  
PRS - Indicates whether the file is protected or not (contains "YES" or "NO")  
RES - contains CHR\$(18), REVERSE ON  
RAS - contains CHR\$(146), REVERSE OFF

### Program Documentation:

110        Set the color code of the screen  
120 - 200 Program heading  
210 - 230 Asks if the names should be listed out.  
         Sets flag FL to 1 and executes the routine at 280-490.  
250 - 270 Input the filename. Asks for new input if the filename if greater than 16 characters.  
280 - 490 Reads the file name from the directory and either displays it (FL=1) or compares it to the desired filename.  
500 - 530 Reads byte 0 (file type) of the file entry of the desired file and stores it in TY. Also, the right half-byte is stored in FT.  
540 - 590 Checks the file type and saves the text in F\$, and checks for invalid file type.  
600 - 610 Checks bit 7 of the file type byte (file closed?) and saves the result in CLS.  
620 - 630 Checks bit 6 of the file type byte (file protected?) and saves the result in PRS.  
640 - 690 Reads the number of allocated blocks in the file from bytes 28 and 29 of the file entry and saves it in BL.



## Anatomy of the 1541 Disk Drive

700 - 730 If it is relative file, the record length is read from byte 21 and saved in RL  
740 - 880 If it is a program file, the starting address of the file is taken from the first data block and stored in SA.  
890 - 980 Free blocks on the disk are calculated by reading the first byte of the track-marked BAM section and added to BF. The allocated blocks are calculated by  $BA = 664 - BF$   
990 -1020 Here the number of side-sector blocks (BS) of a relative file is calculated with the help of the record length (RL) and the number of allocated blocks in the file (RC).  
1040-1230 Here the data can be sent to the screen or the printer as one chooses. The file parameters are shown in REVERSE.  
1240-1280 The parameters of another file can be output.

The program is written for a CBM 64. In spite of this, it can be run without major changes on a VIC 20. Only line 110, where the color of the screen is set, need be changed for the VIC 20.

### BASIC Listing of the Program:

```
100 CLR
110 POKE 53280,2:POKE53281,2:PRINTCHR$(158);CHR$(147);
120 PRINT TAB(6);"=====
130 PRINT TAB(6);"DISPLAY ALL FILE PARAMETERS"
140 PRINT TAB(6);"=====
150 PRINT:PRINT
160 PRINT"WITH THIS PROGRAM, ALL PARAMETERS OF A"
170 PRINT"FILE CAN BE OUTPUT TO THE SCREEN OR TO"
180 PRINT"A PRINTER AT YOUR OPTION."
200 PRINT:PRINT
210 PRINT"LIST FILENAMES (Y/N)?"
220 GETX$:IFX$<>"Y"ANDX$<>"N"THEN220
230 IF X$="Y"THENFL=1:GOSUB280
240 FL=0
250 INPUT"NAME OF THE FILE: ";F$
260 IFLEN(F$)<=16THEN280
270 PRINT"FILENAME TOO LONG!":GOTO250
280 OPEN 15,8,15,"I0":OPEN2,8,2,"#"
290 T=18:S=1
300 PRINT#15,"B-R";2;0;T;S
310 PRINT#15,"B-P";2;0
320 GET#2,X$:IFX$=""THENX$=CHR$(0)
325 T=ASC(X$)
330 GETX$:IFX$=""THENX$=CHR$(0)
340 S=ASC(X$)
350 FORX=0TO7
360 PRINT#15,"B-P";2;X*32+5
370 FFS=""
380 FORY=0TO15
390 GET#2,X$:IFX$=""THENX$=CHR$(0)
```

## Anatomy of the 1541 Disk Drive

```
400 IF ASC(X$)=160THEN430
410 FFS=FF$+X$
420 NEXT Y
430 IFF$=FF$THEN490
440 IFFLTHENPRINTFF$
450 NEXT X
460 IF T=0 THEN 480
470 GOTO 300
480 CLOSE2:CLOSE15
485 IFFL=0THENPRINT"FILENAME NOT FOUND!":GOTO210
490 IFFLTHENRETURN
500 PRINT#15,"B-P";2;X*32+2
510 GET#2,X$:IFX$=""THENX$=CHR$(0)
520 TY=ASC(X$)
530 FT=TYAND15
540 IFFT=0THENFTS="DELETED"
550 IFFT=1THENFTS="SEQUENTIAL"
560 IFFT=2THENFTS="PROGRAM"
570 IFFT=3THENFTS="USER"
580 IFFT=4THENFTS="RELATIVE"
590 IFFT>4THENPRINT"INVALID FILE TYPE!":GOTO200
600 IFTYAND128THENCLS="YES":GOTO620
610 CLS="NO"
620 IFTYAND64THENPRS="YES":GOTO640
630 PRS="NO"
640 PRINT#15,"B-P";2;X*32+30
650 GET#2,X$:IFX$=""THENX$=CHR$(0)
660 LB=ASC(X$)
670 GET#2,X$:IFX$=""THENX$=CHR$(0)
680 HB=ASC(X$)*256
690 BL=LB+HB
700 IFFT<>4THEN740
710 PRINT#15,"B-P";2;X*32+23
720 GET#2,X$:IFX$=""THENX$=CHR$(0)
730 RL=ASC(X$)
740 IFFT<>2THEN890
750 PRINT#15,"B-P";2;X*32+3
760 GET#2,X$:IFX$=""THENX$=CHR$(0)
770 DT=ASC(X$)
780 GET#2,X$:IFX$=""THENX$=CHR$(0)
790 DS=ASC(X$)
800 OPEN3,8,3,"#"
810 PRINT#15,"B-R";3;0;DT;DS
820 PRINT#15,"B-P";3;2
830 GEI#3,X$:IFX$=""THENX$=CHR$(0)
840 LB=ASC(X$)
850 GET#3,X$:IFX$=""THENX$=CHR$(0)
860 HB=ASC(X$)*256
870 SA=LB+HB
880 CLOSE3
890 PRINT#15,"B-R";2;0;18;0
900 BF=0
910 FORI=4TO140STEP4
920 IFI=72THEN960
930 PRINT#15,"B-P";2;I
```

```

940 GET#2,XS:IFXS=""THENXS=CHR$(0)
950 BF=ASC(X$)+BF
960 NEXT
980 BA=664-BF
990 IFFT<>4THEN1040
1010 BS=BL/121:IFBS<>INT(BS)THENBS=INT(BS+1)
1020 RC=INT(((BL-BS)*254)/RL)
1040 PRINTCHR$(147);"SCREEN OR PRINTER (S/P)?"
1050 GETXS:IFXS<>"S"ANDXS<>"P"THEN1050
1060 RES=CHR$(18):RAS=CHR$(146)
1070 IFXS="S"THENOPEN1,3:PRINT#1,CHR$(147)
1080 IFXS="P"THENOPEN1,4
1090 PRINT#1,"FILE PARAMETERS          ";RES;FS;ROS
1100 PRINT#1,"-----"
1110 PRINT#1,"FILE TYPE:                  ";RES;FTS;RAS:PRINT#1
1120 PRINT#1,"FILE CLOSED:                ";RES;CLS;RAS:PRINT#1
1130 PRINT#1,"FILE PROTECTED:              ";RES;PRS;RAS:PRINT#1
1140 PRINT#1,"ALLOCATED BLOCKS:            ";RES;BL;RAS:PRINT#1
1150 IFFT<>4THEN1200
1160 PRINT#1,"RECORD LENGTH:           ";RES;RL;RAS:PRINT#1
1170 PRINT#1,"SIDE-SECTOR BLOCKS:         ";RES;BS;RAS:PRINT#1
1180 PRINT#1,"DATA BLOCKS:                  ";RES;BL-BS;RAS:PRINT#1
1190 PRINT#1,"RECORDS:                      ";RES;RC;RAS:PRINT#1
1200 IFFT=2THENPRINT#1,"START ADDRESS:          ";
    RES;SA;RAS:PRINT#1
1210 PRINT#1,"FREE BLOCKS (DISK):        ";RES;BF;RAS:PRINT#1
1220 PRINT#1,"ALLOCATED BLOCKS (D):";RES;BA;RAS:PRINT#1
1230 CLOSE1
1240 PRINT"MORE (Y/N)?"
1250 CLOSE2:CLOSE15
1260 GETXS:IFXS<>"Y"ANDXS<>"N"THEN1260
1270 IFXS="Y"THEN100

```

#### 4.1.2 Scratch-protect Files - File Protect

As already mentioned, it is possible to protect files on the VIC-1541 diskette and save this information in the directory. A file's type is contained in byte 0 of the file entry. Bit 6 denotes a protected file. If this bit is set to 1, the file can no longer be deleted with the **SCRATCH** command. But because the DOS has no command to set this bit an alternative way must be used to protect a file.

With the following program, you can:

- \* display all files on the disk
- \* protect files
- \* unprotect files
- \* erase files

This program can delete protected files as well as unprotected files. If you wish to delete a protected file,

## Anatomy of the 1541 Disk Drive

you must confirm it. This program is also documented with a variable usage and descriptions so that you can use these techniques in your own programs.

### List of Variables:

DF - Flag, set in the routine "read/search file" if the desired filename is found  
FL - Set if the routine "read/search file" is only to be used for listing files  
FT - Variable for storing the filetype  
T - Track of the actual block of the file entry  
S - Sector of the actual block of the file entry  
TT - Track, in which the file entry block of the desired file is found  
SS - Sector, in which the file entry block of the desired file is found  
FFS - last filename read from the directory  
F\$ - filename to search for

### Program Documentation:

100       Set the screen color  
110 - 230 Program header and option menu  
240 - 260 Read the menu choice and call the appropriate subroutine  
270       Back to the option menu  
280 - 350 Subprogram "list all files"  
310       Erase screen  
320       Set flag FL to list files in the subroutine "read/search file"  
350       Reset the flag and jump back  
360 - 600 Subroutine "protect file"  
390       Call subroutine "input filename"  
400       Call the subroutine "read/search file"  
410 - 450 Test if the file is found  
460 - 480 Read file type and store in FT  
490 - 500 Test if the file is already protected  
510       Protect file (bit 6 to 1)  
520 - 550 Transfer the file type to the buffer and write the block to disk  
560       Close the channel  
570 - 600 Message "File protected" and jump back  
610 - 850 Subroutine "unprotect file"  
640       Call subroutine "input filename"  
650       Call subroutine "read/search file"  
660 - 700 Test if file is found  
710 - 730 Read file type and store in FT  
740 - 750 Test if the file is already unprotected  
760       Unprotect the file (bit 6 to 0)  
770 - 800 Transfer the file type to the buffer and write the block to the disk  
810       Close the file  
820 - 850 End the subroutine

```

860 -1170 Subroutine "erase a file"
      890 Call the subroutine "input filename"
      900 Call the subroutine "read/search file"
910 - 950 Test if the file is found
960 - 980 Read the file type and save in FT
      990 Test if the file is protected
1000-1030 Indicate that the file is protected, with the
      possibility to erase it anyway
1040-1060 Ask if the file should really be erased
      1070 Bit 6 set back, if protected
1080-1110 Transfer the file type to the buffer and write
      the block to the disk
      1120 Initialize the diskette
      1130 Erase the file
1140-1170 End the subroutine
1190-1560 Subroutine "read/search file"
      1220 Open the command and data channels
1230-1240 Read directory and set buffer pointer
1250-1320 Test if the disk contains a write protect. For
      this purpose, the directory is written back to the
      disk unchanged (line 1250). If the disk has a
      write protect tab on it, the error message 26,
      WRITE PROTECT ON will occur.
      1330 Initial values for the track and sector variables
      are set
1340-1350 Read the file entry block and position the buffer
      pointer to the first byte
1360-1390 Read the address of the next file entry block
1400-1530 Loop to read filenames. The names are then either
      listed on the screen or compared to the desired
      filename, based on the value of flag FL
1540-1560 If the variable T (track) contains zero, no more
      file entry blocks follow and the subroutine ends.

```

## BASIC Listing of the Program:

```

100 POKE 53280,2:POKE53281,2:PRINTCHR$(158);CHR$(147);
110 PRINTTAB(8);"=====
120 PRINTTAB(8);"ERASE AND PROTECT FILES"
130 PRINTTAB(8);"=====
140 PRINT:PRINT
150 PRINT"WITH THIS PROGRAM, FILES CAN BE"
160 PRINT"PROTECTED, ERASED, AND UNPROTECTED"
180 PRINT:PRINT
190 PRINTTAB(6);" -1- LIST ALL FILES":PRINT
200 PRINTTAB(6);" -2- PROTECT A FILE":PRINT
210 PRINTTAB(6);" -3- UNPROTECT A FILE":PRINT
220 PRINTTAB(6);" -4- ERASE A FILE":PRINT
230 PRINTTAB(6);" -5- END THE PROGRAM":PRINT
240 GETX$:IFX$=""ORVAL(X$)<1ORVAL(X$)>5THEN240
250 IFVAL(X$)=5THENEND
260 ONVAL(X$)GOSUB280,360,610,860
270 GOTO 100
280 REM -----
290 REM LIST ALL FILES

```

## Anatomy of the 1541 Disk Drive

```
300 REM -----
310 PRINTCHR$(147)
320 FL=1:GOSUB1190
330 PRINT:PRINT"RETURN FOR MORE"
340 INPUTX$
350 FL=0:RETURN
360 REM -----
370 REM PROTECT A FILE
380 REM -----
390 GOSUB1580
400 GOSUB1190
410 IFDF=1THEN460
420 PRINT"FILE NOT FOUND!":PRINT
430 PRINT"RETURN FOR MORE"
440 INPUTX$:CLOSE2:CLOSE15
450 RETURN
460 PRINT#15,"B-P";2;X*32+2
470 GET#2,X$:IFX$=""THENX$=CHR$(0)
480 FT=ASC(X$)
490 IF (FT AND 64)=0 THEN 510
500 PRINT"FILE IS ALREADY PROTECTED!":PRINT:GOTO430
510 FT=(FT OR 64)
520 PRINT#15,"B-P";2;X*32+2
530 PRINT#2,CHR$(FT);
540 PRINT#15,"B-P";2;0
550 PRINT#15,"U2";2;0;TT;SS
560 CLOSE2:CLOSE15
570 PRINT"FILE PROTECTED."
580 PRINT"RETURN FOR MORE"
590 INPUTX$
600 CLOSE2:CLOSE15:RETURN
610 REM -----
620 REM UNPROTECT A FILE
630 REM -----
640 GOSUB1580
650 GOSUB1190
660 IFDF=1THEN710
670 PRINT"FILE NOT FOUND!":PRINT
680 PRINT"RETURN FOR MORE"
690 INPUTX$:CLOSE2:CLOSE15
700 RETURN
710 PRINT#15,"B-P";2;X*32+2
720 GET#2,X$:IFX$=""THENX$=CHR$(0)
730 FT=ASC(X$)
740 IF (FT AND 64)=64THEN760
750 PRINT"FILE IS ALREADY UNPROTECTED!":PRINT:GOTO680
760 FT=(FTAND255-64)
770 PRINT#15,"B-P";2;X*32+2
780 PRINT#2,CHR$(FT);
790 PRINT#15,"B-P";2;0
800 PRINT#15,"U2";2;0;TT;SS
810 CLOSE2:CLOSE15
820 PRINT"FILE UNPROTECTED."
830 PRINT"RETURN FOR MORE"
840 INPUTX$
```

```

850 RETURN
860 REM -----
870 REM ERASE A FILE
880 REM -----
890 GOSUB1580
900 GOSUB1190
910 IFDF=1THEN960
920 PRINT"FILE NOT FOUND!":PRINT
930 PRINT"RETURN FOR MORE"
940 INPUTX$:CLOSE2:CLOSE15
950 RETURN
960 PRINT#15,"B-P";2;X*32+2
970 GET#2,X$:IFX$=""THENX$=CHR$(0)
980 FT=ASC(X$)
990 IF(FT AND 64)=0THEN1040
1000 PRINT"WARNING! FILE IS PROTECTED!"
1010 PRINT"UNPROTECT AND ERASE (Y/N)?"
1020 GETX$:IFX$<>"Y"ANDX$<>"N"THEN1020
1030 IFX$="N"THEN1170
1040 PRINT"ARE YOU SURE (Y/N)?"
1050 GETX$:IFX$<>"Y"ANDX$<>"N"THEN1050
1060 IFX$="N"THEN1170
1070 FT=(FT AND 255-64)
1080 PRINT#15,"B-P";2;X*32+2
1090 PRINT#2,CHR$(FT);
1100 PRINT#15,"B-P";2;0
1110 PRINT#15,"U2";2;0;TT;SS
1120 PRINT#15,"I0"
1130 PRINT#15,"S:"+F$
1140 PRINT"FILE ERASED."
1150 PRINT"RETURN FOR MORE"
1160 INPUTX$
1170 CLOSE2:CLOSE15:RETURN
1180 REM
1190 REM -----
1200 REM READ / SEARCH FILE
1210 REM -----
1220 OPEN15,8,15,"I0":OPEN2,8,2,"#"
1230 PRINT#15,"B-R";2;0;18;0
1240 PRINT#15,"B-P";2;0
1250 PRINT#15,"U2";2;0;18;0
1260 INPUT#15,X1$
1270 IF VAL(X1$)<>26 THEN 1330
1280 PRINT"PLEASE REMOVE THE WRITE PROTECT TAB FROM"
1290 PRINT"THE DISKETTE BEFORE USING THIS PROGRAM."
1300 PRINT"RETURN FOR MORE"
1310 INPUTX$
1320 CLOSE2:CLOSE15:RETURN
1330 T=18:S=1:TT=18:SS=1
1340 PRINT#15,"B-R";2;0;T;S
1345 TT=T:SS=S
1350 PRINT#15,"B-P";2;0
1360 GET#2,X$:IFX$=""THENX$=CHR$(0)
1370 T=ASC(X$)
1380 GET#2,X$:IFX$=""THENX$=CHR$(0)

```

## Anatomy of the 1541 Disk Drive

```
1390 S=ASC(X$)
1400 FORX=0TO7
1410 PRINT#15,"B-P";2;X*32+2
1420 GET#2,X$:IFX$=""THENX$=CHR$(0)
1430 IFASC(X$)=0THEN1530
1440 PRINT#15,"B-P";2;X*32+5
1450 FF$=""
1460 FORY=0TO15
1470 GET#2,X$:IFX$=""THENX$=CHR$(0)
1480 IFASC(X$)=160THEN1500
1490 FF$=FF$+X$
1500 NEXTY
1510 IFFLTHENPRINTFF$:GOTO1530
1520 IFF$=FF$THENDF=1:GOTO1570
1530 NEXTX
1540 IFT<>0THEN1340
1550 CLOSE2:CLOSE15
1560 IFFL=0THENPRINT"FILENAME NOT FOUND!":FORI=1TO2000:
    NEXT
1570 RETURN
1580 REM -----
1590 REM INPUT FILENAME
1600 REM -----
1610 PRINT:PRINT
1620 INPUT"FILENAME: ";F$
1630 IFLEN(F$)<=16THEN1650
1640 PRINT"FILENAME TOO LONG!":GOTO1620
1650 DF=0:FL=0
1660 RETURN
```

This utility program was written for the CBM 64. This version can also be run on the VIC 20. Only line 100 which sets the screen color on the CBM 64 need be changed or ignored. If you value perfect video output, lines 110-230 can also be changed to accommodate the VIC 20's smaller screen size.

### 4.1.3 Backup Program - Copying a Diskette

The VIC 1541 disk drive does not allow disks to be duplicated since it is a single drive, as the double drives permit with the **COPY** or **BACKUP** commands of BASIC 4.0. With the 1541, each program to be copied must be transferred through the computer.

Here's an example of how you might copy a diskette using a single disk drive:

First, the BAM as well as the names and IDs of the disk to be copied are read into the computer. From the information in the BAM, you can determine which blocks of the original diskette are used. In order to save time, only the allocated



blocks are copied. Then a direct access file is opened and the first 169 (as many as will fit in the memory of the Commodore 64) allocated blocks are read. Then the user is asked to put a new diskette in the drive. The new diskette is then formatted with the name and ID of the original diskette. Now the previously read blocks are written to the diskette. The next 169 blocks of the original diskette are read into memory and written out to the destination diskette. This ends after four disk swaps, at which time the entire diskette will have been copied.

The program is written in BASIC except for the portion which reads and writes the direct access file. This part is written in machine language which is considerably faster than a GET# loop in BASIC. Because of the nature of the program, the number of diskette changes is dependent on the free storage in the computer. A VIC 20 with a 16K expansion requires 11 changes of original and destination diskettes.

Here is a time comparison between this program and duplication on a double drive with the same capacity. Our program requires about 20 minutes, while the CBM 4040 does it in about 3 minutes.

Duplicating a diskette with this program is quite simple. You need only follow the messages on the screen to insert the original or destination diskette. The program does the rest for you.

```

100 REM  BACKUP PROGRAM C64 - VIC 1541
110 REM
120 POKE56,23:CLR:GOSUB640
130 OPEN1,8,15
140 DIM B%(35,23),S%(35),Z(7),A$(1)
150 A$(0)="DESTINATION":A$(1)="ORIGINAL":R=1
160 AD=23*256:GOSUB590
170 POKE250,0:POKE251,AD/256
180 GOSUB530:GOSUB290
190 PRINTNS"BLOCKS TO COPY":PRINT
200 T=1:S=0
210 FORI=1TO4:TT=T:SS=S:R=1:IFI=1THEN240
220 IFR=0ANDI=1THENGOSUB450:GOTO240
230 GOSUB590
240 POKE251,AD/256:FORJ=1TO169
250 IFB%(T,S)=0THENGOSUB570
260 S=S+1:IFS=S%(T)THENT=T+1:S=0:IFT=36THENJ=169
270 NEXT:IFRTHENR=0:T=TT:S=SS:GOTO220
280 NEXT:GOTO510
290 T=18:S=0:GOSUB570
300 NS=0:FORT=1TO35:S=0
310 NS=NS+S%(T)-PEEK(AD+4*T)
320 FORJ=1TO3
330 B=PEEK(AD+4*T+J)
340 FORI=0TO7

```

## Anatomy of the 1541 Disk Drive

```
340 FOR I=0 TO 7
350 B%(T,S)=B AND Z(I):S=S+1
360 NEXT I,J
370 FOR S=S%(T) TO 23
380 B%(T,S)=-1 : NEXT S,T
390 FOR I=0 TO 15
400 A=PEEK(AD+144+I)
410 IFA<>160 THEN NS=NS+CHR$(A)
420 NEXT
430 IS=CHR$(PEEK(AD+162))+CHR$(PEEK(AD+163))
440 PRINT NS,IS:RETURN
450 PRINT "PLEASE INSERT NEW DISKETTE"
460 PRINT "AND PRESS RETURN":PRINT:POKE 198,0:CLOSE 2
470 GETAS:IF AS<>CHR$(13) THEN 470
480 PRINT#1,"N0:"NS","I$
490 INPUT#1,A,B$,C,D:IF A THEN PRINT A,"BS","C","D":END
500 GOTO 630
510 CLOSE 2:CLOSE 1:END
520 REM SECTORS PER TRACK
530 FORT=1 TO 35
540 S%(T)=21:IFT>17 THEN S%(T)=19:IFT>24 THEN S%(T)=18:
    IFT>30 THEN S%(T)=17
550 NEXT
560 FOR I=0 TO 7:Z(I)=2↑I:NEXT:RETURN
570 IFR THEN PRINT#1,"U1 2 0":T:S:SYSIN:RETURN
580 PRINT#1,"B-P 2 0":SYSOUT:PRINT#1,"U2 2 0":T:S:RETURN
590 CLOSE 2:PRINT "PLEASE INSERT "AS(R)" DISKETTE."
600 PRINT "AND PRESS RETURN":PRINT:POKE 198,0
610 GETAS:IF AS<>CHR$(13) THEN 610
620 PRINT#1,"I0"
630 OPEN 2,8,2,"#":RETURN
640 FOR I = 828 TO 873 : REM READ MACHINE LANG. PROGRAM
650 READ X : POKE I,X : S=S+X : NEXT
660 DATA 162, 2, 32,198,255,160, 0, 32,207,255,145,250
670 DATA 200,208,248,230,251, 32,204,255, 96,198, 1,162
680 DATA 2, 32,201,255,160, 0,177,250, 32,210,255,200
690 DATA 208,248,230,251, 32,204,255,230, 1, 96
700 IF S<>7312 THEN PRINT "ERROR IN DATA!":END
710 IN=828:OUT=849:RETURN
```

### 4.1.4 Copying Individual Files to another Diskette

The following program permits you to copy individual files from one diskette to another. The files can be programs (PRG), sequential files (SEQ) or user files (USF). Relative files cannot be copied with this program; these can be copied with a BASIC program that reads all data records into a string array and then writes them back again into a new file.

In the first pass, the program reads the complete file into the memory of the Commodore 64. Then the destination

Next the complete file is written on the second disk. The computer has 49 Kbytes for data storage; you can handle up to 196 blocks on the diskette.

For reasons of speed, the reading and writing of the data is performed by a machine language program, which is stored in DATA statements.

The program is suited for copying sequential files as already mentioned, as well as programs of all kinds; the start address (of a machine language program) is not relevant.

```

100 REM  FILE COPIER PROGRAM  C64
110 REM
120 POKE 56,12 : CLR
130 GOSUB 1000
140 INPUT"FILENAME  ";N$
150 PRINT"FILE TYPE ";
160 GETT$:IF T$<>"S"AND T$<>"P"AND T$<>"U" THEN 160
170 PRINTT$:PRINT
180 PRINT"PLEASE INSERT ORIGINAL DISK"
190 PRINT"AND PRESS A KEY":PRINT
200 GETA$:IFA$="" THEN 200
210 OPEN 2,8,2,N$+"."+T$
220 POKE 3,0:POKE 4,12:SYS 866
230 CLOSE 2
240 PRINT"PLEASE INSERT DESTINATION DISK"
250 PRINT"AND PRESS A KEY":PRINT
260 GETA$:IFA$="" THEN 260
270 OPEN 2,8,2,N$+"."+T$+"."W"
280 POKE 3,0:POKE 4,12:SYS 828
290 CLOSE 2 : END
1000 FOR I = 828 TO 898
1010 READ X : POKE I,X : S=S+X : NEXT
1020 DATA 162, 2, 32,201,255,198, 1,160, 0, 56,165, 3
1030 DATA 229, 5,165, 4,229, 6,176, 13,177, 3, 32,210
1040 DATA 255,230, 3,208,236,230, 4,208,232,230, 1, 76
1050 DATA 204,255,162, 2, 32,198,255,160, 0, 32,207,255
1060 DATA 145, 3,230, 3,208, 2,230, 4, 36,144, 80,241
1070 DATA 165, 3,133, 5,165, 4,133, 6, 76,204,255
1080 IF S<>8634 THEN PRINT "ERROR IN DATA !!":END
1090 RETURN

```

#### 4.1.5 Reading the directory from within a program

Sometimes applications programs store user data in a file under a desired name. If you want to use this file again, but you cannot remember the file name, then you have a problem. If this happens, you must exit the program, search for the name in the directory, reload the program and start

## Anatomy of the 1541 Disk Drive

again. To avoid this, you can include a directory listing routine in your program. If you forget the filename, you can display the directory with a function key, for example, without the need to leave the program. Here is a sample of such a routine:

```
100 PRINTCHR$(147);
110 OPEN15,8,15,"IO":OPEN2,8,2,"#"
120 T=18:S=1
130 PRINT#15,"B-R";2;0;T;S
140 PRINT#15,"B-P";2;0
150 GET#2,X$:IFX$=""THENXS=CHR$(0)
160 T=ASC(X$)
170 GET#2,X$:IFX$=""THENXS=CHR$(0)
180 S=ASC(X$)
190 FORX=0TO7
200 PRINT#15,"B-P";2;X*32+5
210 FFS=""
220 FORY=0TO15
230 GET#2,X$:IFX$=""THENXS=CHR$(0)
240 IFASC(X$)=160THEN270
250 FFS=FFS+X$
260 NEXTY
270 IFA=0THENA=1:PRINTFFS;:GOTO290
280 A=0:PRINTTAB(20);FFS
290 NEXTX
300 IFT<>0THEN130
310 CLOSE1:CLOSE2
320 PRINT"RETURN FOR MORE"
330 INPUTXS
340 END:REM IF SUBROUTINE, THEN RETURN HERE
```

In order to select the filename, the directory is printed on the screen. Should this program be used as a subroutine (called with GOSUB) line 340 must contain RETURN instead of END.

We used this routine in the utility programs in sections 4.1.1 and 4.1.2.

## 4.2 The Utility Programs on the TEST/DEMO Disk

There are many 1541 owners that know little about the programs contained on the Test/Demo disk. The main reason is that these programs are largely undocumented. The following descriptions of these programs should help you:

### 4.2.1 DOS 5.1

The DOS 5.1 simplifies the operation of the VIC-1541 DOS. It can run on the VIC-20 or Commodore 64. To load DOS 5.1 on the VIC-20, give the commands

```
LOAD"VIC-20 WEDGE",8
RUN
```

This is the loader for DOS 5.1 for the VIC 20.

If you want to use it on the Commodore 64, give the commands:

```
LOAD"C-64 WEDGE",8
RUN
```

This loads DOS 5.1 into the CBM 64.

What does this DOS 5.1 offer? It allows you to send convenient commands to the 1541 disk drive. If, for example, you want to display the directory on the screen, you use the DOS 5.1 command @\$ or >\$. This does not erase the program in memory.

The individual commands of the DOS 5.1

Command	Function
-----	-----
@\$ or >\$	Display the directory
@V or >V	Same function as "VALIDATE"
@C:... or >C:...	Copy files (COPY)
*file or /file	Load program
@ or >	Read and display error message
@N:... or >N:...	Format a diskette
@I or >I	Initialize the disk
@R:... or >R:...	Rename a file (RENAME)
@S:... or >S:...	Erase a file (SCRATCH)
@#n or >#n	Change disk device to n

## Anatomy of the 1541 Disk Drive

### 4.2.2 COPY/ALL

With the program **COPY/ALL** files can be copied between disk drives with different addresses. A drive must be changed from device address 8 with the program **DISK ADDR CHANGE** before this can occur. After starting the program, the message:

```
disk copy all           jim butterfield  
from unit? 8
```

appears on the screen. Here you give the device address of the disk drive from which you wish to get the files. If this address is 8, just press RETURN. After this you give the corresponding drive number of this unit (always 0 for single drives). In this manner you also give the device address of the destination drive. Once this has occurred, the program asks

```
want to new the output disk  
?n
```

You are being asked if the destination diskette should be formatted. You answer with 'y' (yes) or 'n' (no). Then you can choose the files you want to copy with the wildcard (\*). If all files are supposed to be copied, just give the asterisk.  
Now the program gives the message

```
hold down 'y' or 'n' key to select
```

The program displays the files on the original disk, which you can select with the 'y' key (yes) or 'n' (no). The files by which you pressed 'y' will be copied. If, during the copying process, asterisks (\*\*\*) appear behind the files, it means that an error has occurred. If there is not enough room on the destination disk, "\*\*\*\* output disk full" and "do you have a new one" appears. The remaining files can be put on another formatted diskette. To do this, answer 'y' when ready.

At the the conclusion of the copying process, the number of free blocks on the destination disk is displayed.

### 4.2.3 DISK ADDR CHANGE

With this program, the device address of a disk drive can be changed through software. After starting the program, turn all drives off except for the one you wish to change. Now enter the old and new device addresses.

After this, the address is changed and the other drive can be turned back on.

The following drives can be changed with this program:

2031	DOS V2.6
2040	DOS V1.1
4040	DOS V2.1
4040	DOS V2.7
8050	DOS V2.5
8050	DOS V2.7
8250	DOS V2.7

## 4.2.4 DIR

This is a small help program with the following possibilities:

- d - display the directory on the screen
- > - With this character, a disk command can be given in shortened form (for example, >N:TEST,KN to format a diskette)
- q - exit the program
- s - display the error channel

These possibilities are also found in DOS 5.1, along with other commands.

## 4.2.5 VIEW BAM

With this utility program you can view the usage of diskette blocks on the screen. This table displays the sectors in columns and the tracks in rows. Crosses indicate free blocks and reverse crosses indicate allocated blocks. 'n/a' means that these blocks do not exist on the track.

After outputting the table, the diskette name and the number of free blocks is displayed.

## 4.2.6 CHECK DISK

The utility program CHECK DISK tests every block on the diskette by writing to and reading from it. The current

## **Anatomy of the 1541 Disk Drive**

block and the total number of tested blocks is displayed on the screen.

### **4.2.7 DISPLAY T&S**

If you are interested in the construction of the individual blocks of the disk and want to display these on the screen, this utility program will help you. After starting the program you give the desired track and sector. This will then be sent to the printer or screen. The DISK-MONITOR contained in this book is a easier to use, because it allows you to change blocks and save them again.

### **4.2.8 PERFORMANCE TEST**

This program makes it possible to test the mechanics of the VIC-1541 disk drive. To accomplish this, all the access commands are executed, in the following order:

1. Disk is formatted
2. A file is opened for reading
3. Data are written to this file
4. The file is closed again
5. This file is opened for reading
6. The data are read
7. The file is closed again
8. The file is erased
9. Track 35 is written
10. Track 1 is written
11. Track 35 is read
12. Track 1 is read

After each access of the disk the error channel is displayed. In this manner, it can be established which access of the disk is not executed properly.

When using this program, use only diskettes containing no important data because the entire diskette is erased during the testing.



### 4.3 BASIC-Expansion and Programs for easy Use of the 1541

#### 4.3.1 Input strings of desired length from the disk

Reading data from the disk with the INPUT# statement has one major disadvantage - only data items having fewer than 88 characters can be read. This is because the input buffer of the computer is limited. In addition, not all characters can be read with the INPUT# statement. If a record contains a comma or colon, BASIC views it as a separating character and the remainder of the input is assigned to the next variable. If the INPUT# statement has only one variable, the remainder is ignored and the next INPUT# statement continues reading past the next carriage return (CHRS(13)). The alternative, to read the input with a GET# statement but results in much slower input.

To avoid these disadvantages, we can use a small machine language routine.

We will change the INPUT# statement, so that we can specify the number of characters to be read. To distinguish it from the normal INPUT# statement, we name the command INPUT\*. The syntax looks like this:

```
INPUT* lfn, len, var
```

Lfn is the logical file number of the previously OPENed file, len is the number of characters to be read, and var is the string variable into which the characters are to be read. A program excerpt might look like this:

```
100 OPEN 2,8,2,"FILE"
110 INPUT* 2,100,AS
```

This reads a string of 100 characters from the opened file into AS. This procedure is especially suited for relative files, because a complete record can be read with one command after positioning the record pointer. The partitioning of record into fields can be accomplished with the MIDS function. An elegant method of creating records is described in the next section.

With this procedure it is no longer necessary to end a record with a carriage return. You can especially make use of the maximum record length with relative files:

```
100 OPEN 1,8,15
110 OPEN 2,8,2, "REL-FILE,L,"+CHRS(20)
120 PRINT#1, "P"+CHRS(10)+CHRS(0)+CHRS(1)
130 PRINT#2, "12345678901234567890";
140 PRINT#1, "P"+CHRS(10)+CHRS(0)+CHRS(1)
```

# Anatomy of the 1541 Disk Drive

```
150 INPUT* 2,20,A$
160 PRINT A$
```

```
12345678901234567890
```

Here is the assembler listing for the machine language program. It resides in the cassette buffer just like a loader program in BASIC for the Commodore 64 and VIC 20.

```
110: 033C          ;
          ; INPUT* LFN,LEN,A$
          ;
150: 033C          INPUT      EQU  $85
160: 033C          STAR      EQU  $AC
170: 033C          BASVEC    EQU  $308
180: 033C          CHRGET    EQU  $73
190: 033C          CHRGOT    EQU  CHRGET + 6
          ;
210: 033C          ; C64 VERSION
220: 033C          ;
380: 033C          CHKIN     EQU  $E11E
390: 033C          BASIN     EQU  $E112
400: 033C          CHKCOM    EQU  $AEFD
410: 033C          INTER     EQU  $A7AE
420: 033C          EXECOLD   EQU  $A7E7
430: 033C          INPUTOLD  EQU  $ABBF
440: 033C          FINDVAR   EQU  $B08B
450: 033C          STRRES    EQU  $B475
460: 033C          FRESTR    EQU  $B6A3
470: 033C          GETBYT    EQU  $B79E
          ;
          ; VIC 20 VERSION
          ;
240: 033C          CHKIN     EQU  $E11B
250: 033C          BASIN     EQU  $E10F
260: 033C          CHKCOM    EQU  $CEFD
270: 033C          INTER     EQU  $C7AE
280: 033C          EXECOLD   EQU  $C7E7
290: 033C          INPUTOLD  EQU  $CBBF
300: 033C          FINDVAR   EQU  $D08B
310: 033C          STRRES    EQU  $D475
320: 033C          FRESTR    EQU  $D6A3
330: 033C          GETBYT    EQU  $D79E
          ;
          ; COMMON LABELS
          ;
490: 033C          VARADR     EQU  $49
500: 033C          CLRCH     EQU  $FFCC
510: 033C          PARA      EQU  $61
          ;
530: 033C          ORG       828
540: 033C A9 47          INIT   LDA  #<TEST
550: 033E A0 03          LDY    #>TEST
560: 0340 8D 08 03          STA   BASVEC
570: 0343 8C 09 03          STY   BASVEC+1
```

# Anatomy of the 1541 Disk Drive

```

580: 0346 60                      RTS
600: 0347 20 73 00 ; TEST        JSR  CHRGET
610: 034A C9 85                      CMP  #INPUT
620: 034C F0 06                      BEQ  FOUND
630: 034E 20 79 00                  JSR  CHRGOT
640: 0351 4C E7 A7                  JMP  EXECOLD ; TO THE OLD
                                         ROUTINE
650: 0354 20 73 00 FOUND          JSR  CHRGET
660: 0357 C9 AC                      CMP  #STAR ; NEW INPUT
                                         ROUTINE
670: 0359 F0 06                      BEQ  OKSTAR
680: 035B 20 BF AB                  JSR  INPUTOLD
680: 035E 4C AE A7                  JMP  INTER
690: 0361 20 9B B7 OKSTAR          JSR  GETBYT-3 ; GET FILE
                                         NUMBER
700: 0364 20 1E E1                  JSR  CHKIN
710: 0367 20 FD AE                  JSR  CHKCOM
720: 036A 20 9E B7                  JSR  GETBYT ; LENGTH
730: 036D 8A                      TXA
730: 036E 48                      PHA ; NOTICE
740: 036F 20 FD AE                  JSR  CHKCOM
750: 0372 20 8B B0                  JSR  FINDVAR ; SEARCH FOR
                                         VARIABLE
760: 0375 85 49                      STA  VARADR
760: 0377 84 4A                      STY  VARADR+1
770: 0379 20 A3 B6                  JSR  FRESTR
780: 037C 68                      PLA ; LENGTH
790: 037D 20 75 B4                  JSR  STRRES ; RESERVE PLACE
                                         FOR STRING
800: 0380 A0 02                      LDY  #2
810: 0382 B9 61 00 STORE          LDA  PARA,Y
820: 0385 91 49                      STA  (VARADR),Y
830: 0387 88                      DEY
840: 0388 10 F8                      BPL  STORE
850: 038A C8                      INY ; Y=0
860: 038B 20 12 E1 FETCH          JSR  BASIN
870: 038E 91 62                      STA  (PARA+1),Y
880: 0390 C8                      INY
890: 0391 C4 61                      CPY  PARA
900: 0393 D0 F6                      BNE  FETCH
910: 0395 20 CC FF                  JSR  CLRCH
910: 0398 4C AE A7                  JMP  INTER ;TO INTERPRETER
                                         LOOP

```

Here are the BASIC programs for entering the machine language program for the INPUT\* statement.

INPUT\* , 64 Version

```

100 FOR I = 828 TO 922
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 71,160, 3,141, 8, 3,140, 9, 3, 96, 32

```

## Anatomy of the 1541 Disk Drive

```
130 DATA 115, 0,201,133,240, 6, 32,121, 0, 76,231,167
140 DATA 32,115, 0,201,172,240, 6, 32,191,171, 76,174
150 DATA 167, 32,155,183, 32, 30,225, 32,253,174, 32,158
160 DATA 183,138, 72, 32,253,174, 32,139,176,133, 73,132
170 DATA 74, 32,163,182,104, 32,117,180,160, 2,185, 97
180 DATA 0,145, 73,136, 16,248,200, 32, 18,225,145, 98
190 DATA 200,196, 97,208,246, 32,204,255, 76,174,167
200 IF S <> 11096 THEN PRINT "ERROR IN DATA !!" : END
210 SYS 828 : PRINT "OK."
```

INPUT\* , VIC 20 VERSION

```
100 FOR I = 828 TO 922
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 71,160, 3,141, 8, 3,140, 9, 3, 96, 32
130 DATA 115, 0,201,133,240, 6, 32,121, 0, 76,231,199
140 DATA 32,115, 0,201,172,240, 6, 32,191,203, 76,174
150 DATA 199, 32,155,215, 32, 27,225, 32,253,206, 32,158
160 DATA 215,138, 72, 32,253,206, 32,139,208,133, 73,132
170 DATA 74, 32,163,214,104, 32,117,212,160, 2,185, 97
180 DATA 0,145, 73,136, 16,248,200, 32, 15,225,145, 98
190 DATA 200,196, 97,208,246, 32,204,255, 76,174,199
200 IF S <> 11442 THEN PRINT "ERROR IN DATA !!" : END
210 SYS 828 : PRINT "OK."
```

### 4.3.2 Easy Preparation of Data Records

If you have worked with relative files before, you know that a definite record length must be established. This record is usually divided into several fields which likewise begin at a definite position within the record, and have a set length.

If you create a new record, for example, a separate INPUT statement is generally used for each field. Before the complete record can be written, it must be assembled properly. Each field must be checked for proper length. If it is longer than then the planned length of the corresponding data field, the remainder must be truncated to the proper length. Here are two new BASIC commands that are excellently suited for this task. These new commands are written in machine language and are initialized with a SYS command. You can then use them as any other BASIC commands.

The first command has the name !STR\$ and serves to create a string with the length of the data record.

```
AS = !STR$(100," ")
```

creates a string with 100 blanks and puts it in the variable A\$.

The next command places our data field in the previously created string. For example, if you want to assign the variable N\$ containing the last name as a field of 25 characters at position 1 in the string A\$, our new command looks like this:

```
MID$ (A$,1,25) = N$
```

Here the MID\$ command is used as a so-called pseudo-variable on the left side of the assignment statement. What happens here is as follows:

The variable N\$ replaces the first 25 characters of A\$. If the variable N\$ is longer than 25 characters, only the first 25 characters are replaced and the rest are disregarded. If N\$ is shorter than 25 characters, only as many characters are replaced as N\$ contains. The original characters in A\$ remain (blanks, in our case). That is exactly as we wanted. Now we can program the following:

```
200 INPUT "LAST NAME      "; L$
210 INPUT "FIRST NAME     "; F$
220 INPUT "STREET         "; S$
230 INPUT "CITY           "; C$
240 INPUT "STATE          "; T$
250 INPUT "ZIP CODE       "; Z$
260 A$ = !STR$ (92, " ")
270 MID$ (A$,1,25) = L$
280 MID$ (A$,26,20) = F$
290 MID$ (A$,46,20) = S$
300 MID$ (A$,66,15) = C$
310 MID$ (A$,81,2)  = T$
320 MID$ (A$,83,9)  = Z$
330 PRINT#2, A$
```

Here is the machine language program for the Commodore 64

135:	C800	ORG	\$C800
140:	C800	CHKOPEN	EQU \$AEFA
150:	C800	CHKCLOSE	EQU \$AEF7
160:	C800	CHKCOM	EQU \$AEFD
170:	C800	FRMEVL	EQU \$AD9E
180:	C800	CHKSTR	EQU \$AD8F
190:	C800	FRESTR	EQU \$B6A3
200:	C800	YFAC	EQU \$B3A2
205:	C800	CHRGET	EQU \$73
210:	C800	CHRGOT	EQU CHRGET+6
220:	C800	GETBYT	EQU \$B79B
226:	C800	INTEGER	EQU \$B1AA
229:	C800	DESCRPT	EQU \$64
230:	C800	STRADR	EQU \$62
231:	C800	ADR2	EQU \$FB

# Anatomy of the 1541 Disk Drive

```

232:  C800          ADR1      EQU  $FB+2
233:  C800          LEN1      EQU  3
234:  C800          LEN2      EQU  4
235:  C800          NUMBFR     EQU  5
236:  C800          START      EQU  6
237:  C800          TYPFLAG     EQU  13
238:  C800          STRCODE     EQU  $C4
240:  C800          ILLQUAN     EQU  $B248
241:  C800          SYNTAX      EQU  $AF08
242:  C800          POSCODE     EQU  $B9
243:  C800          VECTOR      EQU  $30A
245:  C800          TEMP        EQU  LEN1
248:  C800 A9 0D      LDA  #<TESTIN
248:  C802 A0 C8      LDY  #>TESTIN
248:  C804 8D 0A 03    STA  VECTOR
248:  C807 8C 0B 03    STY  VECTOR+1
248:  C80A 4C 6B C8    JMP  MIDSTR
250:  C80D A9 00      TESTIN    LDA  #0
250:  C80F 85 0D      STA  TYPFLAG
250:  C811 20 73 00    JSR  CHRGET
251:  C814 C9 21      CMP  #"!"
251:  C816 F0 06      BEQ  TEST2
251:  C818 20 79 00    JSR  CHRGET
251:  C81B 4C 8D AE    JMP  $AE8D
252:  C81E 20 73 00    TEST2    JSR  CHRGET
252:  C821 C9 C4      CMP  #STRCODE
252:  C823 F0 03      BEQ  STRING
253:  C825 4C 08 AF    JMP  SYNTAX

;
;  STRING$ FUNCTION
;
900:  C828 20 73 00    STRING    JSR  CHRGET
900:  C82B 20 FA AE    JSR  CHKOPEN ; OPEN PAREN
910:  C82E 20 9E B7    JSR  GETBYT+3
920:  C831 8A          TXA
920:  C832 48          PHA ; NOTICE LENGTH
930:  C833 20 FD AE    JSR  CHKCOM
940:  C836 20 9E AD    JSR  FRMEVL
950:  C839 24 0D      BIT  TYPFLAG
960:  C83B 30 0C      BMI  STR ; STRING
970:  C83D 20 AA B1    JSR  INTEGER
980:  C840 A5 64      LDA  DESCRIPT ; HIGH BYTE
990:  C842 D0 24      BNE  ILL ; >255
1000: C844 A5 65      LDA  DESCRIPT+1 ; LOW BYTE,
; LENGTH
1010: C846 4C 52 C8    JMP  STR2
1020: C849 20 82 B7    JSR  $B782 ; SETSTR
; TYPFLAG TO
; NUMERIC
1030: C84C F0 1A      BEQ  ILL ; LENGTH 0
1040: C84E A0 00      LDY  #0
1050: C850 B1 22      LDA  ($22),Y ; FIRST CHAR
1060: C852 85 03      STR2     STA  TEMP
1070: C854 68          PLA ; LENGTH
1080: C855 20 7D B4    JSR  $B47D ; FRESTR

```

# Anatomy of the 1541 Disk Drive

```

1090: C858 A8          TAY
1100: C859 F0 07      BEQ  STR3
1110: C85B A5 03      LDA  TEMP
1120: C85D 88          DEY
1120: C85E 91 62      STA  (STRADR),Y ; CREATE
                                STRING
1130: C860 D0 FB      BNE  LOOP
1140: C862 20 CA B4 STR3 JSR  $B4CA ;BRING STRING
                                IN DESCRIPTOR STACK
1150: C865 4C F7 AE      JMP  CHKCLOSE
1160: C868 4C 48 B2 ILL  JMP  ILLQUAN
;
; MID$(STRINGVAR,POS,LEN) = STRING EXP
; MID$(STRINGVAR,POS) = STRING EXP
;
200: C86B          MIDCODE EQU  $CA
210: C86B          EXECUT  EQU  $308 ;VECTOR FOR
                                STATEMENT EXECUTE
240: C86B          EXECOLD EQU  $A7E7
250: C86B          VARNAM  EQU  $45
255: C86B          VARADR  EQU  $49
260: C86B          DESCRPT EQU  $64
270: C86B          TESTSTR EQU  $AD8F
280: C86B          GETVAR  EQU  $B08B
290: C86B          SETSTR  EQU  $AA52
325: C86B          TEST    EQU  $AEFF
330: C86B          GETBYT  EQU  $B79E
355: 0003          ORG     3
360: 0004          LENGTH  DST  1
370: 0005          POSITION  DST  1
372: 0007          VARSTR  DST  2
375: 0007          COMP    EQU  $B2
378: 0007          POINT2  EQU  $50
;
400: C86B A9 76      MIDSTR LDA  #<MIDTEST
410: C86D A0 C8      LDY   #>MIDTEST
420: C86F 8D 08 03   STA   EXECUT
430: C872 8C 09 03   STY   EXECUT+1
440: C875 60          RTS
450: C876 20 73 00 MIDTEST JSR  CHRGET
460: C879 C9 CA      CMP   #MIDCODE ;CODE FOR MIDS
470: C87B F0 06      BEQ   MID ;? YES
480: C87D 20 79 00   JSR  CHRGOT
490: C880 4C E7 A7   JMP   EXECOLD ;EXECUTE
                                NORMAL STATEMENT
500: C883 20 73 00 MID JSR  CHRGET ;NEXT CHAR
505: C886 20 FA AE   JSR  CHKOPEN ;OPEN PAREN
510: C889 20 8B B0   JSR  GETVAR ;GET VAR
520: C88C 85 64      STA   DESCRPT
530: C88E 84 65      STY   DESCRPT+1
535: C890 85 49      STA   VARADR
535: C892 84 4A      STY   VARADR+1
540: C894 20 A3 B6   JSP   FRESTR
545: C897 A0 00      LDY   #0
545: C899 B1 64      LDA   (DESCRPT),Y

```

# Anatomy of the 1541 Disk Drive

545:	C89B	48		PHA		;LENGTH
545:	C89C	F0	2E	BEQ	ILL	
550:	C89E	20	52 AA	JSR	SETSTR	;PUT STRING IN RAM
560:	C8A1	A0	01	LDY	#1	
560:	C8A3	B1	49	LDA	(VARADR),Y	
560:	C8A5	85	05	STA	VARSTR	;VAR ADDR
570:	C8A7	C8		INY		
570:	C8A8	B1	49	LDA	(VARADR),Y	
570:	C8AA	85	06	STA	VARSTR+1	
600:	C8AC	20	FD AE	JSR	CHKCOM	
610:	C8AF	20	9E B7	JSR	GETBYT	;GET POS
620:	C8B2	8A		TXA		
630:	C8B3	F0	17	BEQ	ILL	
650:	C8B5	CA		DEX		
650:	C8B6	86	04	STX	POSITION	
660:	C8B8	20	79 00	JSR	CHRGOT	
660:	C8BB	C9	29	CMP	#")"	;END OF EXPRESSION?
665:	C8BD	D0	04	BNE	NEXT	
665:	C8BF	A9	FF	LDA	#\$FF	;MAX LENGTH
665:	C8C1	D0	0C	BNE	STORE	
670:	C8C3	20	FD AE NEXT	JSR	CHKCOM	
670:	C8C6	20	9E B7	JSR	GETBYT	;GET LEN
680:	C8C9	8A		TXA		
690:	C8CA	D0	03	BNE	*+5	
700:	C8CC	4C	48 B2 ILL	JMP	ILLOUAN	
710:	C8CF	85	03 STORE	STA	LENGTH	
715:	C8D1	68		PLA		
715:	C8D2	38		SEC		
715:	C8D3	E5	04	SBC	POSITION	
717:	C8D5	C5	03	CMP	LENGTH	
717:	C8D7	B0	02	BCS	OK	
717:	C8D9	85	03	STA	LENGTH	
720:	C8DB	20	F7 AE OK	JSR	CHKCLOSE	;CLOSE PAREN
730:	C8DE	A9	B2	LDA	#COMP	
770:	C8E0	20	FF AE	JSR	TEST	
780:	C8E3	20	9E AD	JSR	FRMEVL	;GET EXP
790:	C8E6	20	A3 B6	JSR	FRESTR	
800:	C8E9	A0	02	LDY	#2	
800:	C8EB	B1	64	LDA	(DESCRPT),Y	
800:	C8ED	85	51	STA	POINT2+1	
800:	C8EF	88		DEY		
800:	C8F0	B1	64	LDA	(DESCRPT),Y	
800:	C8F2	85	50	STA	POINT2	
810:	C8F4	88		DEY		
810:	C8F5	B1	64	LDA	(DESCRPT),Y	
820:	C8F7	F0	D3	BEQ	ILL	;0 THEN ERROR
840:	C8F9	C5	03	CMP	LENGTH	
850:	C8FB	B0	02	BCS	OK1	
860:	C8FD	85	03	STA	LENGTH	
870:	C8FF	A5	05 OK1	LDA	VARSTR	
880:	C901	18		CLC		
880:	C902	65	04	ADC	POSITION	
910:	C904	85	05	STA	VARSTR	



```

910:    C906 90 02          BCC  *+4
920:    C908 E6 06          INC  VARSTR+1
940:    C90A A4 03          LDY  LENGTH
950:    C90C 88             LOOP  DEY
950:    C90D B1 50          LDA  (POINT1),Y ;TRANSFER
                                CHARS FROM STRING
960:    C90F 91 05          STA  (VARSTR),Y ;EXP TO VAR
970:    C911 C0 00          CPY   #0
970:    C913 D0 F7          BNE  LOOP
980:    C915 4C AE A7        JMP  $A7AE ;TO INTERPRETER
                                LOOP

```

For those who have no monitor or assembler for the Commodore 64, we have written a loader program in BASIC.

```

100 FOR I = 51200 TO 51479
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 169, 13,160,200,141, 10, 3,140, 11, 3, 76,107
130 DATA 200,169, 0,133, 13, 32,115, 0,201, 33,240, 6
140 DATA 32,121, 0, 76,141,174, 32,115, 0,201,196,240
150 DATA 3, 76, 8,175, 32,115, 0, 32,250,174, 32,158
160 DATA 183,138, 72, 32,253,174, 32,158,173, 36, 13, 48
170 DATA 12, 32,170,177,165,100,208, 36,165,101, 76, 82
180 DATA 200, 32,130,183,240, 26,160, 0,177, 34,133, 3
190 DATA 104, 32,125,180,168,240, 7,165, 3,136,145, 98
200 DATA 208,251, 32,202,180, 76,247,174, 76, 72,178,169
210 DATA 118,160,200,141, 8, 3,140, 9, 3, 96, 32,115
220 DATA 0,201,202,240, 6, 32,121, 0, 76,231,167, 32
230 DATA 115, 0, 32,250,174, 32,139,176,133,100,132,101
240 DATA 133, 73,132, 74, 32,163,182,160, 0,177,100, 72
250 DATA 240, 46, 32, 82,170,160, 1,177, 73,133, 5,200
260 DATA 177, 73,133, 6, 32,253,174, 32,158,183,138,240
270 DATA 23,202,134, 4, 32,121, 0,201, 41,208, 4,169
280 DATA 255,208, 12, 32,253,174, 32,158,183,138,208, 3
290 DATA 76, 72,178,133, 3,104, 56,229, 4,197, 3,176
300 DATA 2,133, 3, 32,247,174,169,178, 32,255,174, 32
310 DATA 158,173, 32,163,182,160, 2,177,100,133, 81,136
320 DATA 177,100,133, 80,136,177,100,240,211,197, 3,176
330 DATA 2,133, 3,165, 5, 24,101, 4,133, 5,144, 2
340 DATA 230, 6,164, 3,136,177, 80,145, 5,192, 0,208
350 DATA 247, 76,174,167
360 IF S <> 31128 THEN PRINT "ERROR IN DATA !!" : END
370 SYS 51200 : PRINT "OK."

```

## 4.3.3 Spooling - Printing Directly from the Disk

If you have a printer connected to your computer in addition to the disk drive, you can use a special characteristic of the the serial bus.

It is possible to send files directly from disk to the

## Anatomy of the 1541 Disk Drive

printer, without the need to transfer it byte by byte with the computer. For example, if you have text saved as a sequential file, and you want to print it on the printer, the following program allows you to do so:

```
100 OPEN 1,4 : REM PRINTER
110 OPEN 2,8,2, "0:TEST" : REM TEXT FILE
120 GET#2, AS : IF ST = 64 THEN 140
130 PRINT#1, AS; : GOTO 120
140 CLOSE 1 : CLOSF 2
150 END
```

Characters are sent from the disk to the printer until the end of file is recognized. Then the two files are closed and the program ended.

The following is done when spooling:

First both files are opened again. Then a command to receive data (Listen) is sent to the printer, while the disk drive receives the command to send data (Talk). Data are sent automatically from the disk to the printer until the end of file is reached. During this time, the computer can be used without interfering with the transfer of data. Only the use of peripheral devices is not possible during this time.

In practice, this is done with a small machine language program. When you want to start printing, you call the program and give the name of the file which you want to send.

### SYS 828, "TEXT"

OPENS the file TEXT on the diskette and sends it to the printer. As soon as the transfer is begun, the computer responds with READY. again and you can use it, as long as no attempt is made to access the serial bus. You can prove that the computer is no longer needed for transfer by pulling out the bus cable to the disk, so that the diskette is connected only to the printer. When the spooling is done, the disk file is still open (the red LED is still lit). You can CLOSE the file and turn the printer off and then back on, and give the SYS command without a filename (the cable to the disk must be attached, of course).

### SYS 828

With same command you can stop a transfer in progress. The machine language program in the form of a loader program for the Commodore 64 and the VIC 20 is found at the end.

Here are some hints for use:

We have successfully used the printer spooling with a Commodore 64 and a VIC 20 with a printer such as the the VIC

1525. Attempts using an Epson printer with a VIC interface as well as the VIC 1526 did not succeed. The serial bus, in contrast with the parallel IEEE bus, appears to be capable of spooling only with limitations. This is why it is necessary to turn the printer off after spooling, because it still blocks the bus. We would be happy if you would inform us of your experience with other printers.

```

;
; 1541 - 64 SPOOL
;
110: 033C      CHRGOT      EQU  $79
130: 033C      LISTEN      EQU  $FFB1
140: 033C      ATNRES      EQU  $EDBE      ;ATN HI
142: 033C      CLOCK      EQU  $EE85      ;CLOCK HI
144: 033C      DATA      EQU  $EE97      ;DATA HI
160: 033C      CLOSE      EQU  $FFC3
170: 033C      CLALL      EQU  $FFE7
175: 033C      SETFIL      EQU  $FFBA
180: 033C      GETNAME     EQU  $E254      ;GET FILENAME
190: 033C      OPEN        EQU  $FFC0
200: 033C      CHKIN       EQU  $FFC6
202: 033C      UNTALK      EQU  $FFAB
204: 033C      UNLISTEN    EQU  $FFAE
230: 033C      FNLEN       EQU  $B7
240: 033C      INDEV       EQU  $99      ;INPUT DEVICE
260: 033C      NMBFLS      EQU  $98      ;NO. OF FILES
280: 033C      ERROR       EQU  $AF08    ;SYNTAX ERROR
;
300: 033C      ORG         828
310: 033C 20 79 00      JSR  CHRGOT      ;MORE CHARS
320: 033F F0 33      BEQ  OFF          ;SPOOL DONE
330: 0341 20 E7 FF      JSR  CLALL
340: 0344 20 54 E2      JSR  GETNAME
350: 0347 A6 B7      LDX  FNLEN
360: 0349 F0 38      BEQ  SYNTAX
370: 034B A9 02      LDA  #2
380: 034D A2 08      LDX  #8
390: 034F A0 02      LDY  #2
400: 0351 20 BA FF      JSR  SETFIL
410: 0354 20 C0 FF      JSR  OPEN        ;OPEN FILE
411: 0357 A9 04      LDA  #4
412: 0359 20 B1 FF      JSR  LISTEN      ;PRINTER
413: 035C 20 BE ED      JSR  ATNRES
420: 035F A2 02      LDX  #2
430: 0361 20 C6 FF      JSR  CHKIN        ;DISK
435: 0364 20 BE ED      JSR  ATNRES
435: 0367 20 85 EE      JSR  CLOCK
435: 036A 20 97 EE      JSR  DATA
510: 036D A9 00      LDA  #0
520: 036F 85 99      STA  INDEV
530: 0371 85 98      STA  NMBFLS
540: 0373 60      RTS
550: 0374 A9 01      LDA  #1
560: 0376 85 98      STA  NMBFLS
OFF

```

## Anatomy of the 1541 Disk Drive

570:	0378	20	AE	FF	JSR	UNLISTEN	
580:	037B	20	AB	FF	JSR	UNTALK	
620:	037E	A9	02		LDA	#2	
630:	0380	4C	C3	FF	JMP	CLOSE	
640:	0383	4C	08	AF	SYNTAX	JMP	ERROR

Here is the BASIC loader program for the Commodore 64.

```
100 FOR I = 828 TO 901
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 32,121, 0,240, 51, 32,231,255, 32, 84,226
130 DATA 166,183,240, 56,169, 2,162, 8,160, 2, 32
140 DATA 186,255, 32,192,255,169, 4, 32,177,255, 32
150 DATA 190,237,162, 2, 32,198,255, 32,190,237, 32
160 DATA 133,238, 32,151,238,169, 0,133,153,133,152
170 DATA 96,169, 1,133,152, 32,174,255, 32,171,255
180 DATA 169, 2, 76,195,255, 76, 8,175
190 IF S <> 9598 THEN PRINT "ERROR IN DATA !!" : END
200 PRINT "OK."
```

For the VIC 20, use the following program:

```
100 FOR I = 828 TO 901
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 32,121, 0,240, 51, 32,231,255, 32, 81,226
130 DATA 166,183,240, 56,169, 2,162, 8,160, 2, 32
140 DATA 186,255, 32,192,255,169, 4, 32,177,255, 32
150 DATA 197,238,162, 2, 32,198,255, 32,197,238, 32
160 DATA 132,239, 32,160,228,169, 0,133,153,133,152
170 DATA 96,169, 1,133,152, 32,174,255, 32,171,255
180 DATA 169, 2, 76,195,255, 76, 8,207
190 IF S <> 9648 THEN PRINT "ERROR IN DATA !!" : END
200 PRINT "OK."
```

#### 4.4 Overlay Technique and Chaining Machine Language Programs

A proven programming technique involves the creation of a menu program which then loads and executes other programs based on the user's choice. There are two variations: preserving or clearing the old variables in the chained program.

It is possible to pass the old variables if the calling program is as large or larger than the chained program. If a program is chained from another program, the pointer to the end of the previous program remains intact, and the new program loads over the old.

In this example, we would get the following result:

```

100 REM PROGRAM 1
110 REM THIS PROGRAM IS LARGER THAN THE SECOND
120 A = 1000
130 LOAD "PROGRAM 2",8

100 REM PROGRAM 2
110 PRINT A

1000

```

If the chained program is larger than the original program, part of the variables are overwritten and contain meaningless values. Moreover, when the variables that the program destroyed are assigned new values, part of the program is also destroyed.

There are two characteristics of passing variables from the previous program that should be noted - for strings and for functions.

Any string variables that are defined as constants enclosed in quotes in the first program, will have a problem. The string variable pointer points to the actual text in the program. If, for example, a string is defined in the first program with the following assignment

```
100 AS = "TEXT"
```

the variable pointer points to the actual text within line number 100. When chaining, the next program does not change this pointer. New text is now at the original location, so the variable has unpredictable contents. We can easily work around this, however. We need only ensure that the text is copied from the program into string storage where text variables are normally stored. You can do this as follows:

```
100 AS = "TEXT" + ""
```

## Anatomy of the 1541 Disk Drive

By concatenating an empty string, you force the contents of the variable to be copied to the string storage area.

Similar considerations apply to function definitions, because here also the pointer points to the definition within the program. Here you must define the function again in the second program, for example:

```
100 DEF FN A(X) = 0.5 * EXP (-X*X)
```

If you want to chain a program, you can continue to use the old variables provided the second program is not longer than the first. If the chained program is longer, and we do not want to preserve the old variables, there is a trick we can use.

We need only set the end-of-program pointer to the end of the new program immediately after loading. This can be done with two POKE commands:

```
POKE 45, PEEK(174) : POKE 46, PEEK (175) : CLR
```

The CLR command is absolutely necessary. This line should be the first line in the chained program. This allows us to chain a large program without transfer of variables. Another, not so elegant method involves writing the load command in the keyboard buffer so the program will automatically be loaded in the direct mode. To do this, we write the LOAD and RUN commands on the screen and fill the keyboard buffer with 'HOME' and carriage returns. An END statement must come after this in the program. The control system then gets the contents of the keyboard buffer in the direct mode and reads the LOAD and RUN commands that control the loading and execution of the program. Because this occurs in the direct mode, the end address of the program is automatically set, the variables are erased and the program is started with the RUN. The disadvantage of this method is that since the LOAD command must appear on the video screen, any display will be destroyed. In practice it looks like this:

```
1000 PRINT CHR$(147)"LOAD"CHR$(34)"PROGRAM 2"CHR$(34)",8"  
1010 PRINT : PRINT : PRINT : PRINT  
1020 PRINT "RUN"  
1030 POKE 631,19 : POKE 632,13 : POKE 633,13  
1040 POKE 634,13 : POKE 635,13 : POKE 636,13  
1050 POKE 198,6 : END
```

You can see that this procedure is more complicated than the previous one; it is only mentioned for the sake of completeness. With the first procedure, only the LOAD command need be programmed in line 1000:

```
1000 LOAD "PROGRAM 2",8
```

There is another technique for chaining machine language programs.

If a machine language program is to be used from a BASIC program, it must usually be loaded at the beginning of the BASIC program. You must take note of two things:

First of all, you must make sure that the machine language program loads to a specific place in memory. If you load a program without additional parameters, the control system treats it as a BASIC program and loads it at the starting address of the BASIC RAM, generally at 2049 (Commodore 64). Machine language programs can only be run, however, when they are loaded at the address for which they were written. This absolute loading can be accomplished by adding the secondary address 1:

```
LOAD "MACH-PRG",8,1
```

But remember that when loading a program from within another program, BASIC attempts to RUN the program from the beginning. This leads to an endless loop when loading machine language programs, because the operating system thinks that a new BASIC program has been chained:

```
100 LOAD "MACH-PRG",8,1
```

Here we can make use of the fact that the variables are preserved when chaining. If we program the following, we have reached our goal:

```
100 IF A=0 THEN A=1 : LOAD "MACH-PRG",8,1
110 ...
```

When the program is started with RUN, A has the value zero and the assignment after the THEN is executed, A contains the value 1 and the machine language program is then LOADED. When the program begins again after LOADING the program MACH-PRG, A has the value 1 so the next line is executed.

The procedure is similar if you have several machine language programs to load.

```
100 IF A=0 THEN A=1 : LOAD "PROG 1",8,1
110 IF A=1 THEN A=2 : LOAD "PROG 2",8,1
120 IF A=2 THEN A=3 : LOAD "PROG 3",8,1
130 ....
```

The first time through, PROG 1 will be loaded, the next time, PROG 2, and so on. Once all the programs are loaded, execution continues with line 130.

### 4.5 Merge - Appending BASIC Programs

Certainly you have thought about the possibility of combining two separate BASIC programs into one. Without further details this is not possible, because loading the second program would overwrite the first. With the knowledge of how BASIC programs are stored in memory and on the diskette, you can develop a simple procedure to accomplish this task.

BASIC programs are stored in memory as follows:

NL NH	pointer to the next program line, lo hi
LL LH	line number, lo hi
XX YY ZZ	..... tokenized BASIC statements
00	end-of-line marker

At the end of the program are two additional zero bytes:  
00 00     a total of 3 zero bytes

Programs are also saved in this format. Where the program starts and ends lies in two pointers in page zero:

```
PRINT PEEK(43) + 256 * PEEK(44)
```

gives the start of BASIC, 2049 for the Commodore 64,

```
PRINT PEEK(45) + 256 * PEEK(46)
```

points to the byte behind the three zero bytes.

Because a program is always loaded at the start of BASIC, contained in the pointer at 43/44, one can cause a second program to load at the end of the first. In practice, we must proceed as follows:

First we load the first program into memory.

```
LOAD "PROGRAM 1",8
```

Now get the value of the ending address of the program.

```
A = PEEK(45) + 256 * PEEK(46)
```

This value is decremented by two so that the two zero bytes at the end of the program are known.

```
A = A - 2
```

Now, note the original value of the start of BASIC.

```
PRINT PEEK(43), PEEK(44)
```

Next, set the start of BASIC to this value.



POKE, A AND 255 : POKE 44, A / 256

Now, LOAD the second program.

LOAD "PROGRAM 2",8

If you set the start of BASIC back to the original value, 1 and 8 for the Commodore 64 (as shown above with the PRINT commands), you have the complete program in memory and can view it with LIST or save it with SAVE.

POKE 43,1 : POKE 44,8

The following should be noted when using this method:

The appended program may contain only line numbers that are greater than the largest line number of the first program. Otherwise these line numbers can never be accessed with GOTO or GOSUB and the proper program order cannot be guaranteed.

This procedure is especially well suited for constructing a subroutine library for often used routines, so they need not be typed in each time. It will work out best if you reserve specific line numbers for the subroutines, such as 20000-25000, 25000-30000, and so on. If you want to merge several programs in this manner, you must first load the program with smallest line numbers, and then the program with the next highest numbers, etc.

### 4.6 Disk Monitor for the Commodore 64 and VIC 20

In this section we present a very useful tool for working with your disk drive, allowing you to load, display, modify, and save desired blocks on the diskette.

For reasons of speed, the program is written entirely in machine language. The following commands are supported:

- \* Read a block from the disk
- \* Write a block to the disk
- \* Display a block on the screen
- \* Edit a block on the screen
- \* Send disk commands
- \* Display disk error messages
- \* Return to BASIC

The program announces its execution (automatically by the BASIC load program) with

```
DISK-MONITOR V1.0  
>
```

and waits for your input. If you enter '@', the error message from the disk will be displayed, for example

```
00, ok,00,00
```

If you want to send a command to the disk, enter an '@' followed by the command.  
You can initialize a diskette with

```
>@I
```

You can send complete disk commands in this manner, that you would otherwise send with

```
OPEN 15,8,15  
PRINT# 15,"command"  
CLOSE 15
```

For example, you can erase files, format disks, and so on.

The most important function of the disk monitor is the direct access of any block on the diskette. For this, you use the commands R and W. R stands for READ and reads a desired block, W stands for WRITE and writes a block to the disk. You need only specify the track and sector you want to read. These must be given in hexadecimal, exactly as the output is given on the screen. If, for example, you want to read track 18, sector 1 (the first directory block), enter the following command:

```
>R 12 01
```

Each input must be given as a two-digit hex number, separated from each other with a blank.

In order to display the block, use the command **M**. We receive the following output:

```
DISK-MONITOR V1.0
>M
>:00 12 04 82 11 01 47 52 41 .....GRA
>:08 46 49 4B 20 41 49 44 2E FIX AID.
>:10 53 52 43 A0 A0 00 00 00 SRC ...
>:18 00 00 00 00 00 00 15 00 .....
>:20 00 00 82 13 00 48 50 4C .....HPL
>:28 4F 54 2E 53 52 43 A0 A0 OT.SRC
>:30 A0 A0 A0 A0 A0 00 00 00 ...
>:38 00 00 00 00 00 00 05 00 .....
>:40 00 00 82 13 03 56 50 4C .....VPL
>:48 4F 54 2E 53 52 43 A0 A0 OT.SRC
>:50 A0 A0 A0 A0 A0 00 00 00 ...
>:58 00 00 00 00 00 00 09 00 .....
>:60 00 00 82 13 09 4D 45 4D .....MEM
>:68 2E 53 52 43 A0 A0 A0 A0 .SRC
>:70 A0 A0 A0 A0 A0 00 00 00 ...
>:78 00 00 00 00 00 00 06 00 .....
>:80 00 00 82 13 08 4D 45 4D .....MEM
>:88 2E 4F 42 4A A0 A0 A0 A0 .OBJ
etc.
```

Let's take a closer look at the output. The first hex number after the colon gives the address of the following 8 bytes in the block, 00 indicates the first byte in the block (the numbering goes from 00 to FF (0-255)). 8 bytes follow the address (4 on the VIC 20). In the right half are the corresponding ASCII characters. If the code is not printable (\$00 to \$1F and \$80 to \$9F), a period is printed. When you give the command **M**, as above, the entire block is displayed. Because the block does not fit on the screen completely, it is possible to display only part of it. You can give an address range that you would like to display. If you only want to see the first half, enter:

```
>M 00 7F
```

The second half with:

```
>M 80 FF
```

With the VIC 20, you can view quarters of the block. If you now wish to change some data, you simply move the cursor to the corresponding place, overwrite the appropriate byte, and press RETURN. The new value is now stored and the right half is updated with the proper ASCII character.

To write the modified block back to the diskette, you use the command **W**. Here also you must give the track and sector

## Anatomy of the 1541 Disk Drive

numbers in hexadecimal.

>W 12 01

writes the block back to track 18, sector 1, from where we had read the block previously.

If you want to get back to BASIC, enter **X** and the computer will respond with **READY..** If you then want to use the disk monitor again, you need not load it again. Just type **SYS 49152** for the C64 or **SYS 6690** for the VIC 20.

### A warning:

Be sure to make a copy of any diskette that you work with in this way. Should you make an error when editing or writing a block, you can destroy important information on the disk so that it can no longer be used in the normal manner. You should make it a rule to only work with a copy.

Here you find an assembler listing of the program. After this are the BASIC loader programs for the Commodore 64 and VIC 20.

```

;
; disk monitor vic 20 / cbm 64
;
190:  C000      PROMPT      EQU  ">"
200:  C000      NCMD5      EQU  6          ;NUMBER OF
                                         COMMANDS
210:  C000      INPUT      EQU  $FFCF
220:  C000      TALK       EQU  $FFB4
230:  C000      SECTALK    EQU  $FF96
240:  C000      IEEEIN     EQU  $FFA5
250:  C000      UNTALK     EQU  $FFAB
260:  C000      LISTEN     EQU  $FFB1
270:  C000      SECLIST    EQU  $FF93
280:  C000      IEEEOUT    EQU  $FFA8
290:  C000      UNLIST     EQU  $FFAE
300:  C000      WRITE      EQU  $FFD2
310:  C000      OPEN       EQU  $FFC0
320:  C000      CLOSE     EQU  $FFC3
330:  C000      SETPAR     EQU  $FFBA
340:  C000      SETNAM     EQU  $FFBD
350:  C000      CHKIN      EQU  $FFC6
360:  C000      CKOUT      EQU  $FFC9
370:  C000      CLRCH      EQU  $FFCC
380:  C000      CR         EQU  13
390:  C000      QUOTE      EQU  $22
400:  C000      QUOTFLG    EQU  $D4
410:  0200      ORG        EQU  $200      ;BASIC INPUT
                                         BUFFER
420:  0201      SAVX       BYT  0
430:  0202      WRAP      BYT  0
440:  0203      BAD       BYT  0
```

# Anatomy of the 1541 Disk Drive

```

450: 0204      FROM      BYT  0
460: 0205      TO        BYT  0
470: 0205      STATUS    EQU  $90
480: 0205      SA         EQU  $B9      ;SECONDARY
                                         ADDRESS
490: 0205      FA         EQU  $BA      ;DEVICE #
500: 0205      FNADR      EQU  $BB      ;FILENAME ADR
510: 0205      FNLEN      EQU  $B7      ;LEN OF
                                         FILENAME
520: 0205      TMPC       EQU  $97
610: C000      COUNT      EQU  8      ;# OF BYTES PER LINE
620: C000      READY      EQU  $E37B    ;$E467 FOR VIC
630: C000 A2 00      INIT   LDX  #0
640: C002 BD 85 C2    MSGOUT LDA MESSAGE,X
650: C005 20 D2 FF    JSR  WRITE
660: C008 E8          INX
670: C009 E0 12      CPX  #ASCDMP-MESSAGE
680: C00B D0 F5      BNE  MSGOUT
690: C00D A2 0D      START  LDX  #CR
700: C00F A9 3E      LDA  #PROMPT
710: C011 20 EB C0    JSR  WRTWHR
710: C014 A9 00      LDA  #0
710: C016 8D 01 02    STA  WRAP
720: C019 20 33 C1 ST1 JSR  RDOC ;READ INPUT LINE
730: C01C C9 3E      CMP  #PROMPT
740: C01E F0 F9      BEQ  ST1
750: C020 C9 20      CMP  #" " ;READ OVER BLANK
760: C022 F0 F5      BEQ  ST1
770: C024 A2 05      S0     LDX  #NCMDS-1 ;COMPARE WITH
                                         COMMAND TABLE
780: C026 DD 6A C0 S1    CMP  CMDS,X
790: C029 D0 0C      BNE  S2
800: C02B 8E 00 02    STX  SAVX ;# OF CMDS IN TABLE
840: C02E BD 70 C0    LDA  ADRH,X
850: C031 48          PHA
                                         ;JUMP ADDR TO
                                         STACK
860: C032 BD 76 C0    LDA  ADRL,X
870: C035 48          PHA
880: C036 60          RTS
890: C037 CA          S2     DEX
900: C038 10 EC      BPL  S1 ;LOOP OF ALL CMDS
910: C03A 4C 0D C0    JMP  START
;
; SUBROUTINE TO DISPLAY
; THE DISK CONTENTS
960: C03D 85 97      DM     STA  TMPC
970: C03F 20 62 C0 DM1   JSR  SPACE
980: C042 B9 E0 C2     LDA  BUFFER,Y ;GET BYTE FROM
                                         BUFFER
990: C045 20 DC C0     JSR  WROB
1000: C048 C8          INY
1000: C049 D0 03      BNE  DM2
1000: C04B FE 01 02    INC  WRAP
1010: C04E C6 97      DM2    DEC  TMPC
1020: C050 D0 ED      BNE  DM1

```

# Anatomy of the 1541 Disk Drive

```

1030:  C052 60                      RTS
                                ; READ BYTES AND WRITE TO MEMORY
1060:  C053 20 FE C0 BYT          JSR RDOB
1070:  C056 90 03                BCC BY3 ;BLANK?
1080:  C058 99 E0 C2              STA BUFFER,Y ;WRITE BYTE IN
                                BUFFER

1090:  C05B C8                    BY3    INY
1100:  C05C C6 97                DEC TMPC
1110:  C05E 60                    RTS
1120:  C05F 20 62 C0 SPAC2        JSR SPACE
1130:  C062 A9 20                  LDA #" "
1140:  C064 2C                      BYT    $2C
1150:  C065 A9 0D                  LDA #CR
1160:  C067 4C D2 FF              JMP WRITE

                                ;
                                ; COMMAND AND ADDRESS TABLE
1190:  C06A 3A                    CMDS    ASC ':' ;EDIT MEM CONTENTS
1200:  C06B 57                    ASC 'W' ;WRITE BLOCK
1210:  C06C 52                    ASC 'R' ;READ BLOCK
1220:  C06D 4D                    ASC 'M' ;DISLPAY BYTES
1230:  C06E 40                    ASC '@' ;DISK COMMAND
1240:  C06F 58                    ASC 'X' ;EXIT
1250:  C070 C0                    ADRH     EQU >ALTM-1
1260:  C071 C1                    EQU >DIRECT-1
1270:  C072 C1                    EQU >DIRECT-1
1280:  C073 C0                    EQU >DSPLYM-1
1290:  C074 C1                    EQU >DISK-1
1300:  C075 E3                    EQU >READY-1
1310:  C076 C0                    ADRL     EQU <ALTM-1
1320:  C077 90                    EQU <DIRECT-1
1330:  C078 90                    EQU <DIRECT-1
1340:  C079 7B                    EQU <DSPLYM-1
1350:  C07A 3E                    EQU <DISK-1
1360:  C07B 7A                    EQU <READY-1
1370:  C07C A0 00                DSPLYM   LDY #0
1380:  C07E 8C 03 02            STY FROM
1370:  C081 88                    DEY
1370:  C082 8C 04 02            STY TO
1370:  C085 20 CF FF            JSR INPUT
1370:  C088 C9 0D                CMP #CR
1370:  C08A F0 17                BEQ DSP1
1380:  C08C 20 FE C0            JSR RDOB ;READ START
                                ADDRESS

1390:  C08F 90 12                BCC DSP1
1400:  C091 8D 03 02            STA FROM
1410:  C094 20 CF FF            JSR INPUT
1410:  C097 C9 0D                CMP #CR
1410:  C099 F0 08                BEQ DSP1
1420:  C09B 20 FE C0            JSR RDOB ;READ END ADR
1430:  C09E 90 03                BCC DSP1
1440:  C0A0 8D 04 02            STA TO
1450:  C0A3 AC 03 02 DSP1        LDY TO
1460:  C0A6 20 C6 C2 DSP2        JSR TESTEND
1470:  C0A9 20 D6 C2            JSR ALTRIT
1470:  C0AC 98                    TYA

```

# Anatomy of the 1541 Disk Drive

```

1480:  C0AD 20 DC C0          JSR  WROB          ;ADDRESS
1490:  C0B0 20 62 C0          JSR  SPACE          ;OMIT FOR VIC
1500:  C0B3 A9 08            LDA  #COUNT        ;8 OR 4
1510:  C0B5 20 3D C0          JSR  DM           ;DISPLAY
1520:  C0B8 20 97 C2          JSR  ASCDMP        ;ASCII DUMP
1530:  C0BB 4C A6 C0          JMP  DSP2         ;ABS JUMP
1550:  C0BE 4C 0D C0 BEQ$1     JMP  START
                                ;EDIT MEMORY; READ ADDRESS AND DATA
1570:  C0C1 20 FE C0 ALTM     JSR  RDOB          ;READ ADDR
1580:  C0C4 90 F8            BCC  BEQ$1
1590:  C0C6 A8                TAY
1600:  C0C7 A9 08            LDA  #COUNT        ;# OF BYTES
1610:  C0C9 85 97            STA  TMPC
1610:  C0CB 20 33 C1          JSR  RDOC          ;OMIT FOR VIC
1620:  C0CE 20 33 C1 A5       JSR  RDOC
1620:  C0D1 20 53 C0          JSR  BYT
1630:  C0D4 D0 F8            BNE  A5
1640:  C0D6 20 97 C2          JSR  ASCDMP
1650:  C0D9 4C 0D C0          JMP  START

                                ;
                                ;WRITE BYTE AS HEX NUMBER
1710:  C0DC 48                WROB  PHA
1720:  C0DD 4A                LSR  A
1730:  C0DE 4A                LSR  A
1740:  C0DF 4A                LSR  A
1750:  C0E0 4A                LSR  A
1760:  C0E1 20 F4 C0          JSR  ASCII          ;CONVERT TO
                                                ASCII
1770:  C0E4 AA                TAX
1780:  C0E5 68                PLA
1790:  C0E6 29 0F            AND  #$0F
1800:  C0E8 20 F4 C0          JSR  ASCII

                                ; WRITE CHARACTERS IN X AND A
1820:  C0EB 48                WRTWHR PHA
1830:  C0EC 8A                TXA
1840:  C0ED 20 D2 FF          JSR  WRITE
1850:  C0F0 68                PLA
1860:  C0F1 4C D2 FF          JMP  WRITE
1870:  C0F4 18                ASCII CLC
1880:  C0F5 69 F6            ADC  #$F6
1890:  C0F7 90 02            BCC  ASC1
1900:  C0F9 69 06            ADC  #6
1910:  C0FB 69 3A            ASC1  ADC  #$3A
1920:  C0FD 60                RTS

                                ; READ HEX BYTE AND PUT IN A
1950:  C0FE A9 00            RDOB  LDA  #0
1960:  C100 8D 02 02          STA  BAD          ;READ NEXT CHAR
1970:  C103 20 33 C1          JSR  RDOC
1980:  C106 C9 20            RDOB1 CMP  #'
1990:  C108 D0 09            BNE  RDOB2
2000:  C10A 20 33 C1          JSR  RDOC          ;READ NEXT CHAR
2010:  C10D C9 20            CMP  #'
2020:  C10F D0 0F            BNE  RDOB3
2030:  C111 18                CLC          ;CY=0
2040:  C112 60                RTS

```

# Anatomy of the 1541 Disk Drive

```

2050: C113 20 28 C1 RDOB2      JSR  HEXIT
2060: C116 0A                  ASL  A
2070: C117 0A                  ASL  A
2080: C118 0A                  ASL  A
2090: C119 0A                  ASL  A
2100: C11A 8D 02 02           STA  BAD
2110: C11D 20 33 C1          JSR  RDOC
2120: C120 20 28 C1 RDOB3     JSR  HEXIT
2130: C123 0D 02 02          ORA  BAD
2140: C126 38                SEC          ;CY=1
2150: C127 60                RTS
2160: C128 C9 3A            HEXIT    CMP  $$3A
2170: C12A 08                PHP
2180: C12B 29 0F            AND  $$0F
2190: C12D 28                PLP
2200: C12E 90 02            BCC  HEX09      ;0-9
2210: C130 69 08            ADC  #8          ;PLUS 9 (C-1)
2220: C132 60                RTS
2230: C133 20 CF FF RDOC     JSR  INPUT      ;READ CHAR
2240: C136 C9 0D            CMP  #CR        ;CR?
2250: C138 D0 F8            BNE  HEX09      ;NO, RETURN
2260: C13A 68                PLA
2270: C13B 68                PLA
2280: C13C 4C 0D C0         JMP  START

;
;
; DOS SUPPORT
2320: C13F 20 CF FF DISK     JSR  INPUT
2330: C142 C9 0D            CMP  #CR
2340: C144 D0 27            BNE  DSKCMD      ;DISK COMMAND
2350: C146 A9 00            LDA  #0
2350: C148 85 90            STA  STATUS      ;ERASE STATUS
2360: C14A 20 65 C0         JSR  CRLF
2370: C14D A9 08            LDA  #8
2380: C14F 85 BA            STA  FA          ;DISK ADDR
2390: C151 20 B4 FF         JSR  TALK
2400: C154 A9 6F            LDA  #15+$60     ;SA 15
2410: C156 85 B9            STA  SA
2420: C158 20 96 FF         JSR  SECTALK     ;SEC ADDR
2430: C15B 20 A5 FF ERRIN   JSR  IEEEIN
2440: C15E 24 90            BIT  STATUS
2440: C160 70 05            BVS  ENDDSK
2450: C162 20 D2 FF         JSR  WRITE
2460: C165 D0 F4            BNE  ERRIN
2470: C167 20 AB FF ENDDSK  JSR  UNTALK
2480: C16A 4C 0D C0         JMP  START
2490: C16D C9 24            DSKCMD    CMP  #'$
2500: C16F F0 1D            REQ  ERR1      ;CATALOG
2510: C171 48                PHA
2510: C172 A9 08            LDA  #8
2520: C174 85 BA            STA  FA
2530: C176 20 B1 FF         JSR  LISTEN
2540: C179 A9 6F            LDA  #15+$60
2550: C17B 85 B9            STA  SA
2560: C17D 20 93 FF         JSR  SECLIST

```



# Anatomy of the 1541 Disk Drive

2560:	C180	68			PLA	
2570:	C181	20	A8	FF	CMDOUT	JSR IEEEOUT
2580:	C184	20	CF	FF		JSR INPUT
2590:	C187	C9	0D			CMP #CR
2600:	C189	D0	F6			BNE CMDOUT
2610:	C18B	20	AE	FF		JSR UNLIST
2630:	C18E	4C	0D	C0	ERR1	JMP START
2640:	C191	20	33	C1	DIRECT	JSR RDOC
2640:	C194	20	FE	C0		JSR RDOB ;READ TRACK
2650:	C197	90	F5			BCC ERR1
2660:	C199	8D	27	C2		STA TRACK
2670:	C19C	20	33	C1		JSR RDOC
2670:	C19F	20	FE	C0		JSR RDOB
2680:	C1A2	90	EA			BCC ERR1
2690:	C1A4	8D	2A	C2		STA SECTOR
2690:	C1A7	20	49	C2		JSR OPNDIR
2690:	C1AA	AD	00	02		LDA SAVX
2690:	C1AD	C9	01			CMP #1
2690:	C1AF	F0	1F			BEO DIRWRITE
2700:	C1B1	A9	31			LDA #'1
2710:	C1B3	20	ED	C1		JSR SENDCMD ;SEND BLOCK READ COMMAND
2720:	C1B6	A2	0D			LDX #13
2730:	C1B8	20	C6	FF		JSR CHKIN
2740:	C1BB	A2	00			LDX #0
2750:	C1BD	20	CF	FF	DIRIN	JSR INPUT
2760:	C1C0	9D	E0	C2		STA BUFFER,X
2770:	C1C3	E8				INX
2770:	C1C4	D0	F7			BNE DIRIN
2780:	C1C6	20	CC	FF		JSR CLRCH
2790:	C1C9	20	6E	C2	ENDDIR	JSR CLSDIR
2790:	C1CC	4C	0D	C0		JMP START
2800:	C1CF	20	2C	C2	DIRWRITE	JSR BUFNT ;SET BUFFER POINTER
2810:	C1D2	A2	0D			LDX #13
2820:	C1D4	20	C9	FF		JSR CKOUT
2830:	C1D7	A2	00			LDX #0
2840:	C1D9	BD	E0	C2	DIROUT	LDA BUFFER,X
2850:	C1DC	20	D2	FF		JSR WRITE
2860:	C1DF	E8				INX
2860:	C1E0	D0	F7			BNE DIROUT
2870:	C1E2	20	CC	FF		JSR CLRCH
2880:	C1E5	A9	32			LDA #'2
2890:	C1E7	20	ED	C1		JSR SENDCMD ;SEND BLOCK WRITE COMMAND
2900:	C1EA	4C	C9	C1		JMP ENDDIR
2910:	C1ED	8D	20	C2	SENDCMD	STA CMDSTR+1
2910:	C1F0	A2	0F			LDX #15
2920:	C1F2	AD	27	C2		LDA TRACK
2920:	C1F5	20	78	C2		JSR NUMBASC
2920:	C1F8	8E	27	C2		STX TRACK
2920:	C1FB	8D	28	C2		STA TRACK+1
2930:	C1FE	AD	2A	C2		LDA SECTOR
2930:	C201	20	78	C2		JSR NUMBASC
2930:	C204	8E	2A	C2		STX SECTOR

# Anatomy of the 1541 Disk Drive

2930:	C207	8D	2B	C2		STA	SECTOR+1	
2940:	C20A	A2	0F			LDX	#15	
2940:	C20C	20	C9	FF		JSR	CKOUT	
2950:	C20F	A2	00			LDX	#0	
2960:	C211	BD	1F	C1	COMDOUT	LDA	CMDSTR,X	
2970:	C214	20	D2	FF		JSR	WRITE	
2980:	C217	E8				INX		
2980:	C218	E0	0D			CPX	#BUFPNT-CMDSTR	
2990:	C21A	D0	F5			BNE	COMDOUT	
3000:	C21C	4C	CC	FF		JMP	CLRCH	
3010:	C21F	55	31	3A	CMDSTR	ASC	'U1:13 0 '	
			31	33	20			
			30	20				
3020:	C227	00	00	20	TRACK	BYT	0,0,\$20	
3030:	C22A	00	00		SECTOR	BYT	0,0	
3040:	C22C	A2	0F		BUFPNT	LDX	#15	
3050:	C22E	20	C9	FF		JSR	CKOUT	
3060:	C231	A2	00			LDX	#0	
3070:	C233	BD	41	C2	PNTOUT	LDA	BUFTXT,X	
3080:	C236	20	D2	FF		JSR	WRITE	
3090:	C239	E8				INX		
3090:	C23A	E0	08			CPX	#OPNDIR-BUFTXT	
3100:	C23C	D0	F5			BNE	PNTOUT	
3110:	C23E	4C	CC	FF		JMP	CLRCH	
3120:	C241	42	2D	50	BUFTXT	ASC	'B-P 13 0'	
			20	31	33			
			20	30				
3130:	C249	A9	0F		OPNDIR	LDA	#15	
3130:	C24B	A8				TAY		
3140:	C24C	A2	08			LDX	#8	
3150:	C24E	20	BA	FF		JSR	SETPAR	
3160:	C251	A9	00			LDA	#0	
3170:	C253	20	BD	FF		JSR	SETNAM	
3180:	C256	20	C0	FF		JSR	OPEN	
3190:	C259	A9	0D			LDA	#13	
3190:	C25B	A8				TAY		
3200:	C25C	A2	08			LDX	#8	
3210:	C25E	20	BA	FF		JSR	SETPAR	
3220:	C261	A9	01			LDA	#1	
3230:	C263	A2	6D			LDX	#<DADR	
3240:	C265	A0	C2			LDY	#>DADR	
3250:	C267	20	BD	FF		JSR	SETNAM	
3260:	C26A	4C	C0	FF		JMP	OPEN	
3270:	C26D	23			DADR	.BYT	'#	
3280:	C26E	A9	0D		CLSDIR	LDA	#13	
3290:	C270	20	C3	FF		JSR	CLOSE	
3300:	C273	A9	0F			LDA	#15	
3310:	C275	4C	C3	FF		JMP	CLOSE	
3230:	C278	A2	30		NUMBASC	LDX	#'0	;HEX # TO AS
3330:	C27A	38				SEC		
3340:	C27B	E9	0A		NUMB1	SBC	#10	
3350:	C27D	90	03			BCC	NUMB2	
3360:	C27F	E8				INX		
3370:	C280	B0	F9			BCS	NUMB1	
3380:	C282	69	3A		NUMB2	ADC	#\$3B	;'9' + 1

# Anatomy of the 1541 Disk Drive

```

3390:  C284 60          RTS
3400:  C285 0D          EQU  CR
3410:  C286 44 49 53    MESSAGE  ASC  'DISK-MONITOR V1.0'
        4B 2D 4D
        4F 4E 49
        54 4F 52
        20 56 31
        2E 30
3430:  C297 98          ASCDMP   TYA
3440:  C298 38          SEC
3440:  C299 E9 08          SBC  #COUNT
3440:  C29B A8          TAY
3450:  C29C 20 62 C0      JSR  SPACE
3460:  C29F A9 12          LDA  #18          ;RVS ON
3470:  C2A1 20 D2 FF      JSR  WRITE
3480:  C2A4 A2 08          LDX  #COUNT
3490:  C2A6 B9 E0 C2 AC2   LDA  BUFFER,Y
3500:  C2A9 29 7F          AND  #$7F
3510:  C2AB C9 20          CMP  #'
3520:  C2AD B0 04          BCS  AC3
3530:  C2AF A9 2E          LDA  #'
3540:  C2B1 D0 03          BNE  AC4
3550:  C2B3 B9 E0 C2 AC3   LDA  BUFFER,Y
3560:  C2B6 20 D2 FF AC4   JSR  WRITE
3570:  C2B9 A9 00          LDA  #0
3570:  C2BB 85 D4          STA  QUOTFLG
3580:  C3BD C8          INY
3580:  C2BE CA          DEX
3590:  C2BF D0 E5          BNE  AC2
3600:  C2C1 A9 92          LDA  #146          ;RVS OFF
3610:  C2C3 4C D2 FF      JMP  WRITE
3620:  C2C6 AD 01 02 TESTEND LDA  WRAP
3620:  C2C9 D0 06          BNE  ENDEND
3630:  C2CB CC 04 02          CPY  TO
3640:  C2CE B0 01          BCS  ENDEND
3650:  C2D0 60          RTS
3660:  C2D1 68          ENDEND  PLA
3660:  C2D2 68          PLA
3660:  C2D3 4C 0D C0      JMP  START
3670:  C2D6 20 65 C0 ALTRIT JSR  CRLF
3680:  C2D9 A9 3A          LDA  #'
3690:  C2DB A2 3E          LDX  #PROMPT
3700:  C2DD 4C EB C0      JMP  WRTWHR
3730:  C2E0          BUFFER  DST  256 ;256 BYTE BUFFER
                                   FOR BLOCK

```

Here is the BASIC program for entering the disk monitor if you do not have an assembler.

# Anatomy of the 1541 Disk Drive

## DISK-MONITOR, C64 VERSION

```

100 FOR I = 49152 TO 49887
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 162, 0,189,133,194, 32,210,255,232,224, 18,208
130 DATA 245,162, 13,169, 62, 32,235,192,169, 0,141, 1
140 DATA 2, 32, 51,193,201, 62,240,249,201, 32,240,245
150 DATA 162, 5,221,106,192,208, 12,142, 0, 2,189,112
160 DATA 192, 72,189,118,192, 72, 96,202, 16,236, 76, 13
170 DATA 192,133,151, 32, 98,192,185,224,194, 32,220,192
180 DATA 200,208, 3,238, 1, 2,198,151,208,237, 96, 32
190 DATA 254,192,144, 3,153,224,194,200,198,151, 96, 32
200 DATA 98,192,169, 32, 44,169, 13, 76,210,255, 58, 87
210 DATA 82, 77, 64, 88,192,193,193,192,193,227,192,144
220 DATA 144,123, 62,122,160, 0,140, 3, 2,136,140, 4
230 DATA 2, 32,207,255,201, 13,240, 23, 32,254,192,144
240 DATA 18,141, 3, 2, 32,207,255,201, 13,240, 8, 32
250 DATA 254,192,144, 3,141, 4, 2,172, 3, 2, 32,198
260 DATA 194, 32,214,194,152, 32,220,192, 32, 98,192,169
270 DATA 8, 32, 61,192, 32,151,194, 76,166,192, 76, 13
280 DATA 192, 32,254,192,144,248,168,169, 8,133,151, 32
290 DATA 51,193, 32, 51,193, 32, 83,192,208,248, 32,151
300 DATA 194, 76, 13,192, 72, 74, 74, 74, 74, 32,244,192
310 DATA 170,104, 41, 15, 32,244,192, 72,138, 32,210,255
320 DATA 104, 76,210,255, 24,105,246,144, 2,105, 6,105
330 DATA 58, 96,169, 0,141, 2, 2, 32, 51,193,201, 32
340 DATA 208, 9, 32, 51,193,201, 32,208, 15, 24, 96, 32
350 DATA 40,193, 10, 10, 10, 10,141, 2, 2, 32, 51,193
360 DATA 32, 40,193, 13, 2, 2, 56, 96,201, 58, 8, 41
370 DATA 15, 40,144, 2,105, 8, 96, 32,207,255,201, 13
380 DATA 208,248,104,104, 76, 13,192, 32,207,255,201, 13
390 DATA 208, 39,169, 0,133,144, 32,101,192,169, 8,133
400 DATA 186, 32,180,255,169,111,133,185, 32,150,255, 32
410 DATA 165,255, 36,144,112, 5, 32,210,255,208,244, 32
420 DATA 171,255, 76, 13,192,201, 36,240, 29, 72,169, 8
430 DATA 133,186, 32,177,255,169,111,133,185, 32,147,255
440 DATA 104, 32,168,255, 32,207,255,201, 13,208,246, 32
450 DATA 174,255, 76, 13,192, 32, 51,193, 32,254,192,144,234
460 DATA 245,141, 39,194, 32, 51,193, 32,254,192,144,234
470 DATA 141, 42,194, 32, 73,194,173, 0, 2,201, 1,240
480 DATA 30,169, 49, 32,237,193,162, 13, 32,198,255,162
490 DATA 0, 32,207,255,157,224,194,232,208,247, 32,204
500 DATA 255, 32,110,194, 76, 13,192, 32, 44,194,162, 13
510 DATA 32,201,255,162, 0,189,224,194, 32,210,255,232
520 DATA 208,247, 32,204,255,169, 50, 32,237,193, 76,201
530 DATA 193,141, 32,194,162, 15,173, 39,194, 32,120,194
540 DATA 142, 39,194,141, 40,194,173, 42,194, 32,120,194
550 DATA 142, 42,194,141, 43,194,162, 15, 32,201,255,162
560 DATA 0,189, 31,194, 32,210,255,232,224, 13,208,245
570 DATA 76,204,255, 85, 49, 58, 49, 51, 32, 48, 32, 0
580 DATA 0, 32, 0, 0,162, 15, 32,201,255,162, 0,189
590 DATA 65,194, 32,210,255,232,224, 8,208,245, 76,204
600 DATA 255, 66, 45, 80, 32, 49, 51, 32, 48,169, 15,168
610 DATA 162, 8, 32,186,255,169, 0, 32,189,255, 32,192

```

# Anatomy of the 1541 Disk Drive

```

620 DATA 255,169, 13,168,162, 8, 32,186,255,169, 1,162
630 DATA 109,160,194, 32,189,255, 76,192,255, 35,169, 13
640 DATA 32,195,255,169, 15, 76,195,255,162, 48, 56,233
650 DATA 10,144, 3,232,176,249,105, 58, 96, 13, 68, 73
660 DATA 83, 75, 45, 77, 79, 78, 73, 84, 79, 82, 32, 86
670 DATA 49, 46, 48,152, 56,233, 8,168, 32, 98,192,169
680 DATA 18, 32,210,255,162, 8,185,224,194, 41,127,201
690 DATA 32,176, 4,169, 46,208, 3,185,224,194, 32,210
700 DATA 255,169, 0,133,212,200,202,208,229,169,146, 76
710 DATA 210,255,173, 1, 2,208, 6,204, 4, 2,176, 1
720 DATA 96,104,104, 76, 13,192, 32,101,192,169, 58,162
730 DATA 62, 76,235,192
740 IF S <> 90444 THEN PRINT "ERROR IN DATA !!": END
750 SYS 49152

```

## DISK-MONITOR, VIC 20 VERSION

In order to allow this program to be run on the VIC 20, it was split into two parts. Enter each program separately, saving the first under the name "DOS LOADER.1" and second under "DOS LOADER.2". To load the disk monitor, load the first program and start it with RUN. If all data are correct, the second program will automatically be loaded and the disk monitor started.

```

100 POKE 55, 6690 AND 255 : POKE 56, 6690 / 256 : CLR
105 FOR I = 6690 TO 7056 : REM DOS LOADER.1
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 162, 0,189,164, 28, 32,210,255,232,224, 18,208
130 DATA 245,162, 13,169, 62, 32, 7, 27,169, 0,141, 1
140 DATA 2, 32, 79, 27,201, 62,240,249,201, 32,240,245
150 DATA 162, 5,221,140, 26,208, 12,142, 0, 2,189,146
160 DATA 26, 72,189,152, 26, 72, 96,202, 16,236, 76, 47
170 DATA 26,133,151, 32,132, 26,185, 0, 29, 32,248, 26
180 DATA 200,208, 3,238, 1, 2,198,151,208,237, 96, 32
190 DATA 26, 27,144, 3,153, 0, 29,200,198,151, 96, 32
200 DATA 132, 26,169, 32, 44,169, 13, 76,210,255, 58, 87
210 DATA 82, 77, 64, 88, 26, 27, 27, 26, 27,228,223,175
220 DATA 175,157, 90,102,160, 0,140, 3, 2,136,140, 4
230 DATA 2, 32,207,255,201, 13,240, 23, 32, 26, 27,144
240 DATA 18,141, 3, 2, 32,207,255,201, 13,240, 8, 32
250 DATA 26, 27,144, 3,141, 4, 2,172, 3, 2, 32,229
260 DATA 28, 32,245, 28,152, 32,248, 26,169, 4, 32, 95
270 DATA 26, 32,182, 28, 76,200, 26, 76, 47, 26, 32, 26
280 DATA 27,144,248,168,169, 4,133,151, 32, 79, 27, 32
290 DATA 117, 26,208,248, 32,182, 28, 76, 47, 26, 72, 74
300 DATA 74, 74, 74, 32, 16, 27,170,104, 41, 15, 32, 16
310 DATA 27, 72,138, 32,210,255,104, 76,210,255, 24,105
320 DATA 246,144, 2,105, 6,105, 58, 96,169, 0,141, 2
330 DATA 2, 32, 79, 27,201, 32,208, 9, 32, 79, 27,201
340 DATA 32,208, 15, 24, 96, 32, 68, 27, 10, 10, 10, 10
350 DATA 141, 2, 2, 32, 79, 27, 32, 68, 27, 13, 2, 2
360 DATA 56, 96,201, 58, 8, 41, 15, 40,144, 2,105, 8

```

# Anatomy of the 1541 Disk Drive

```

370 DATA 96, 32,207,255,201, 13,208,248,104,104, 76, 47
380 DATA 26, 32,207,255,201, 13,208, 39,169, 0,133,144
390 DATA 32,135, 26,169, 8,133,186, 32,180,255,169,111
400 DATA 133,185, 32,150,255, 32,165,255, 36,144,112, 5
410 DATA 32,210,255,208,244, 32,171,255, 76, 47, 26,201
420 DATA 36,240, 29, 72,169, 8,133
430 IF S <> 35614 THEN PRINT "ERROR IN DATA !!" : END
440 LOAD "DOS LOADER.2",8

```

```

100 CLR : FOR I = 7057 TO 7422 : REM DOS LOADER.2
110 READ X : POKE I,X : S=S+X : NEXT
120 DATA 186, 32,177,255,169,111,133,185, 32,147,255,104
130 DATA 32,168,255, 32,207,255,201, 13,208,246, 32,174
140 DATA 255, 76, 47, 26, 76, 47, 26, 32, 79, 27, 32, 26
150 DATA 27,144,245,141, 70, 28, 32, 79, 27, 32, 26, 27
160 DATA 144,234,141, 73, 28, 32,104, 28,173, 0, 2,201
170 DATA 1,240, 30,169, 49, 32, 12, 28,162, 13, 32,198
180 DATA 255,162, 0, 32,207,255,157, 0, 29,232,208,247
190 DATA 32,204,255, 32,141, 28, 76, 47, 26, 32, 75, 28
200 DATA 162, 13, 32,201,255,162, 0,189, 0, 29, 32,210
210 DATA 255,232,208,247, 32,204,255,169, 50, 32, 12, 28
220 DATA 76,232, 27,141, 63, 28,162, 15,173, 70, 28, 32
230 DATA 151, 28,142, 70, 28,141, 71, 28,173, 73, 28, 32
240 DATA 151, 28,142, 73, 28,141, 74, 28,162, 15, 32,201
250 DATA 255,162, 0,189, 62, 28, 32,210,255,232,224, 13
260 DATA 208,245, 76,204,255, 85, 49, 58, 49, 51, 32, 48
270 DATA 32, 0, 0, 32, 0, 0,162, 15, 32,201,255,162
280 DATA 0,189, 96, 28, 32,210,255,232,224, 8,208,245
290 DATA 76,204,255, 66, 45, 80, 32, 49, 51, 32, 48,169
300 DATA 15,168,162, 8, 32,186,255,169, 0, 32,189,255
310 DATA 32,192,255,169, 13,168,162, 8, 32,186,255,169
320 DATA 1,162,140,160, 28, 32,189,255, 76,192,255, 35
330 DATA 169, 13, 32,195,255,169, 15, 76,195,255,162, 48
340 DATA 56,233, 10,144, 3,232,176,249,105, 58, 96, 13
350 DATA 68, 73, 83, 75, 45, 77, 79, 78, 73, 84, 79, 82
360 DATA 32, 86, 49, 46, 48,152, 56,233, 4,168, 32,132
370 DATA 26,169, 18, 32,210,255,162, 4,185, 0, 29, 41
380 DATA 127,201, 32,176, 4,169, 46,208, 3,185, 0, 29
390 DATA 32,210,255,169, 0,133,212,200,202,208,229,169
400 DATA 146, 76,210,255,173, 1, 2,208, 6,204, 4, 2
410 DATA 176, 1, 96,104,104, 76, 47, 26, 32,135, 26,169
420 DATA 58,162, 62, 76, 7, 27
430 IF S <> 39496 THEN PRINT "ERROR IN DATA !!" : END
440 SYS 6690

```

## Chapter 5: The Larger CBM Disks

### 5.1 IEEE-Bus and Serial Bus

Standard Commodore 64's and VIC 20's have a serial bus over which they communicate with peripheral devices such as the VIC 1541 disk drive as well as printers and plotters.

The principle of the bus makes it possible to chain peripherals. Each device has its own device address over which one can communicate with it. The standard address of the disk is 8, a printer is usually 4. The device address is identical to the primary address in the OPEN command. For instance,

```
OPEN 1,4
```

opens a channel to the printer. In order to open several disk files at once, another address, the secondary address, serves to distinguish them. The disk has 16 secondary addresses at its disposal, from 0 to 15. Three secondary addresses are reserved, while the other 13 can be freely used:

Secondary address 0 is used for loading programs.

Secondary address 1 is used for saving programs.

Secondary address 15 is the command and error channel.

The secondary addresses from 2 to 14 can be used for opening files as desired.

The transfer of information between the Commodore 64 and the VIC 1541 occurs serially over this bus. Serial means that the data is sent a bit at a time over just one wire. Data within the computer and disk drive are stored and manipulated in 8 bit groups called bytes. When a byte is sent serially, each individual bit must be sent over the data line. In order that the sender and receiver can stay in step, a so-called 'handshake' line is needed. If we look at the pin-out of the serial bus, we find 6 wires:

Pin	Function
1	SQ IN
2	ground
3	ATN
4	CLK
5	DATA
6	RESET

If the computer wants to send data to the disk drive, the

## Anatomy of the 1541 Disk Drive

ATN (attention) line is set. When this signal is high, all peripherals on the bus stop their work and read the next byte. The data is sent bit-wise over the DATA line. So that the receivers know when the next bit comes, a signal is also sent along the CLCK (clock) line. This transmitted byte is the device address. If this value does not correspond with the device address of a receiving peripheral, the rest of the data is ignored. If, however, the device is addressed, a secondary address may be transmitted. Along with the device address (0 to 31), the device is informed by means of the other three bits whether it is supposed to receive data (LISTEN) or send data (TALK). Following this, data is sent from the computer or from the addressed device.

The RESET line resets all attached devices when the computer is turned on. Over the SRQ IN (service request) line, peripheral devices can inform the bus controller (in our case, the computer), if data is ready, for example. However, this line is not checked by the control system in the Commodore computers.

If one wants to attach several disk drives to the same computer, each must have a different peripheral address. If this is done only occasionally, the program **DISK ADDR CHANGE** can be used, as described in section 4.2.3. The new address (9 for example), remains only until the device is turned off. If the change should be permanent, it can be changed with DIP switches in the drive.

The principle of transfer of data over the IEEE 488 bus is similar to the serial bus function. The important difference is that the data is transmitted over 8 data lines in parallel, not serial. In addition, more handshake lines are needed, so the IEEE bus requires a 24-line cable. The main advantage of the IEEE 488 bus is its ability to transmit a byte at a time, resulting in a higher rate of transfer. Measurements indicate that the IEEE-bus is about 5 times faster than the serial bus: 1.8 Kbyte/second vs. 0.4 Kbyte/second. Loading a 10K program with the VIC 1541 takes about 25 seconds; on the identical 2031, it takes less than 6. This reason alone is enough to warrant outfitting your computer with an IEEE bus.

At the same time, it is possible to use all the other peripherals that the large CBM computers can access.



## 5.2 Comparison of all CBM Disk Drives

In the following table you find the technical data of all CBM disk drives compared.

### The Technical Data of all Commodore Disk Drives

Model	1541	2031	4040	8050	8250
DOS version(s)	2.6	2.6	2.1/ 2.7	2.5/ 2.7	2.7
Drives	1	1	2	2	2
Heads per drive	1	1	1	1	2
Storage capacity	170 K	170 K	340 K	1.05 M	2.12 M
Sequential files	168 K	168 K	168 K	521 K	1.05 M
Relative files	167 K	167 K	167 K	183 K/ 518 K	1,04 M
Buffer storage (KB)	2	2	4	4	4
Tracks	35	35	35	77	77
Sectors per track	17-21	17-21	17-21	23-29	23-29
Bytes per block	256	256	256	256	256
Free blocks	664	664	1328	4104	8266
Directory and BAM (track)	18	18	18	38/39	38/39
Directory entries	144	144	144	224	224
Transfer rate (KB/s)					
internal	40	40	40	40	40
over ser./IEEE bus	0.4	1.8	1.8	1.8	1.8
Access time (ms)					
Track to track	30	30	30	5	5
Average time	360	360	360	125	125
Revolutions/minute	300	300	300	300	300

### Overview of the "large" CBM drives

The VIC 1541 disk drive has the smallest storage capacity of the CBM disks, but it is also the only drive that can be connected directly to the Commodore 64 and VIC 20 over the serial bus.

The functions, construction, and operation are identical to those of the CBM 2031 drive. The only difference from the VIC 1541 is the parallel IEEE bus instead of the serial bus.

## Anatomy of the 1541 Disk Drive

This results in an increase in the transfer rate to the computer of a factor of 5. To connect a Commodore 64 or VIC 20, one needs an IEEE interface, as with all other CBM drives. The storage format of the 2031 is compatible to the 1541; both have 170K per disk. Diskettes can be written with one device and read with the other. This is true for the next drive in the line, the CBM 4040. The 4040 is a double drive with 170K per drive.

The advantage of a double drive lies not only in the increased storage capacity, but also in the ability to transfer data from drive to drive. It is possible to copy complete programs and files using the existing 1541 command.

```
OPEN 1,8,15, "C1:TEST=0:TEST" or
```

```
COPY "TEST",D0 TO "TEST",D1
```

copies the file **TEST** from drive 0 to drive 1 with the same name. In this manner one can concatenate several files on different drives. The most important capability of double drives is the ability to duplicate entire diskettes. This is accomplished by a command from the computer; the drive automatically formats the disk and then makes a track by track copy from one drive to the other. The command to do this is worded:

```
OPEN 1,8,15, "D1=0" or
```

```
BACKUP D0 TO D1
```

The process takes less than 3 minutes on the 4040; during this time the computer may be used since the disk drive performs the entire operation by itself.

The two other CBM drives, the CBM 8050 and the CBM 8250 operate in double density (77 tracks). Disks written with the 1541 or 4040 are not compatible with the 8050/8250. Programs and data can be copied with the **COPY/ALL** program, which transfers from one format to another. This is the reason these drives have greater storage capacity: 1 MB for the 8050 and 2 MB for the 8250. The doubled capacity of the 8250 comes about because both sides of the disk are used (double-sided); it has two reads/write heads per drive. In order to be able to use the whole capacity for relative files (see section 3.4), a so-called 'super side-sector' was introduced, which contains pointers to 127 groups of 6 side-sector blocks each. Through this, a relative file can (theoretically) hold 23 MB of data. These drives can be connected to a Commodore 64 or VIC 20 over an IEEE bus, so that these computers can also access several megabytes.

An additional advantage of the large CBM drives is their larger buffer storage. It is possible to have more files open simultaneously than on the VIC 1541. Up to 5 sequential

files or 3 relative files may be open at any one time, as well as combinations of the two, of course.

With the 8050/8250 format, tracks 38 and 39 are used for the BAM and directory. The disk name and format marker are in track 39 sector 0.

```
>:00 26 00 43 00 00 00 43 42 &.C...CB
>:08 4E 20 38 30 35 30 A0 A0 M 8050
>:10 A0 A0 A0 A0 A0 A0 A0 A0
>:18 30 31 A0 32 43 A0 A0 A0 01 2C
```

The track/sector pointer to the first BAM block (track 38 sector 0) is in bytes 0 and 1. Byte 2 contains the format marker 'C'. Bytes 3 through 5 are unused. The disk name is in 6 to 21, filled with shifted spaces, in our case **CBM 8050**. Bytes 24 and 25 contain the id '01', while bytes 26 and 27 contain the DOS format **2C**.

The BAM no longer occupies just one block, but is dispersed over track 38; sectors 0 and 3 are used in the 8050, the 8250 used sectors 6 and 9 in addition. Because more sectors are use per track, the BAM entry for each track has been enlarged to 5 bytes. The first byte still contains the number of free sectors per track and the following bytes contain the bit model of the free and allocated sectors (0 = sector allocated, 1 = sector free). Here we have the contents of track 38 sector 0

```
>:00 26 03 43 00 01 33 1D FF
>:08 FF FF 1F 1D FF FF FF 1F
>:10 1D FF FF FF 1F 1D FF FF
>:18 FF 1F 1D FF FF FF 1F 1D
>:20 FF FF FF 1F 1D FF FF FF
>:28 1F 1D FF FF FF 1F 1D FF
>:30 FF FF 1F 1D FF FF FF 1F
>:38 1D FF FF FF 1F 1D FF FF
>:40 FF 1F 1D FF FF FF 1F 1D
>:48 FF FF FF 1F 1D FF FF FF
>:50 1F 1D FF FF FF 1F 1D FF
>:58 FF FF 1F 1D FF FF FF 1F
>:60 1D FF FF FF 1F 1D FF FF
>:68 FF 1F 1D FF FF FF 1F 1D
>:70 FF FF FF 1F 1D FF FF FF
>:78 1F 1D FF FF FF 1F 1D FF
>:80 FF FF 1F 1D FF FF FF 1F
>:88 1D FF FF FF 1F 1D FF FF
>:90 FF 1F 1D FF FF FF 1F 1D
>:98 FF FF FF 1F 1D FF FF FF
>:A0 1F 1D FF FF FF 1F 1D FF
>:A8 FF FF 1F 18 FC F3 EF 1F
>:B0 00 00 00 00 00 00 00 00
>:B8 00 00 00 00 00 00 00 0F
>:C0 F4 93 46 1A 18 6C FB FF
>:C8 1F 00 00 00 00 00 00 00
```

# Anatomy of the 1541 Disk Drive

```
>:D0 00 00 00 00 00 00 00 00
>:D8 05 00 00 4D 04 1B FF FF
>:E0 FF 07 1B FF FF FF 07 1B
>:E8 FF FF FF 07 1B FF FF FF
>:F0 07 1B FF FF FF 07 1B FF
>:F8 FF FF 07 1B FF FF FF 07
```

Bytes 0 and 1 point to the next BAM block, track 38 sector 3. Byte 2 contains the format marker 'C' again. The track numbers belonging to this BAM section are in bytes 4 and 5; here tracks 1 through 51. At position 6 we find the 5 byte entry for each track. The next BAM block is constructed similarly. The last BAM block always points to the first directory block: track 39 sector 1.

Four BAM blocks are needed for the 8250: track 38 sector 0 contains the tracks 1 to 51, track 38 sector 3 contains 52 to 100, track 38 sector 6 contains track 101 through 150 and track 38 sector 9 pertains to tracks 151 to 154.

The directory track, track 39, contains 28 free blocks; up to  $28 \times 8 = 224$  directory entries can be stored, in contrast to 144 for the 1541/4040. The construction of the directory is alike for all formats. The following table illustrates the track/sector layout:

	1541 / 4040	8050 / 8250	
Tracks	1 - 17 : 0 - 20	1 - 39 : 0 - 28	sectors
	18- 24 : 0 - 18	40 - 53 : 0 - 26	
	25- 30 : 0 - 17	54 - 64 : 0 - 24	
	31- 35 : 0 - 16	65 - 77 : 0 - 22	
		8250 only	
		78 -116 : 0 - 28	
		117 -130 : 0 - 26	
		131 -141 : 0 - 24	
		142 -154 : 0 - 22	
Blocks	683	2083 : 4186	
Free blocks	664	2052 : 4133	

OTHER BOOKS AVAILABLE:

**The Anatomy of the Commodore 64** - is our insider's guide to your favorite computer. This book is a must for those of you who want to delve deep into your micro. This 300+ page book is full of information covering all aspects of the '64. Includes fully commented listing of the ROMs so you can investigate the mysteries of the BASIC interpreter, kernal and operating system. It offers numerous examples of machine language programming and several samples that make your programming sessions more enjoyable and useful.

ISBN# 0-916439-00-3      Available now:                      \$19.95

**The Anatomy of the 1541 Disk Drive** - unravels the mysteries of working the the Commodore 1541 disk drive. This 320+ page book starts by explaining program, sequential and relative files. It covers the direct access commands, diskette structure, DOS operation and utilities. The fully commented ROM listings are presented for the real "hackers". Includes listings for several useful utilities including BACKUP, COPY, FILE PROTECTOR, DIRECTORY. This is the authoritative source for 1541 disk drive information.

ISBN# 0-916439-01-1      Available now:                      \$19.95

**Tricks & Tips for the Commodore 64** - presents a collection of easy-to-use programming techniques and hints. Chapters cover advanced graphics, easy data entry, enhancements for advanced BASIC, CP/M, connecting to the outside world and more. Other tips include sorting, variable dumps, and POKES that do tricks. All-in-all a solid set of useful features.

ISBN# 0-916439-03-8      Available June 29th:                      \$19.95

**Machine Language Book of the Commodore 64** - is aimed at the owner who wants to progress beyond BASIC and write faster, more memory efficient programs in machine language. The book is specifically geared to the Commodore 64. Learn all of the 6510 instructions as they apply to the '64. Access ROM routines, I/O, extend BASIC, more. Included are listings of three full length programs: an ASSEMBLER; a DISASSEMBLER; and an amazing 6510 SIMULATOR so the reader can "see" the operation of the '64.

ISBN# 0-916439-02-X      Available now:                      \$14.95  
Optional program diskette:      \$14.95

OTHER TITLES COMING SOON!!!

---

# THE ANATOMY OF THE 1541 DISK DRIVE

---

This in depth guide for the Commodore 1541 disk drive owner unravels the mysteries of using the 1541 for programs, sequential and **relative** files with plenty of working examples. This book includes several useful utilities — DISK MONITOR, FILE PROTECTOR, BACKUP, MERGE and more. The Anatomy also discusses the internals of the **Disk Operating System** with the complete fully commented ROM listings.

ISBN 0-916439-01-1

YOU CAN COUNT ON  
**Abacus**  
Software



P.O. Box 7211 Grand Rapids, MI 49510 616/241-5510